AMSTRAD

GUIDE DU GRAPHISME

JAMES WYNORD




AMSTRAD
GUIDE DU GRAPHISME



JAMES WYNFORD



Paris • Berkeley • Düsseldorf

Traduction française de Nellie Saumont

SYBEX n'est lié à aucun constructeur.

Tous les efforts ont été faits pour fournir dans ce livre une information complète et exacte. Néanmoins, SYBEX n'assume de responsabilités, ni pour son utilisation, ni pour les contrefaçons de brevets ou atteintes aux droits de tierces personnes qui pourraient résulter de cette utilisation.

ISBN Version originale : 0-7447-0027-2
ISBN Version française : 2-7361-0141-3

S O M M A I R E

INTRODUCTION

Ce livre est une introduction à quelques-unes des techniques de programmation graphique qui peuvent être exploitées sur l'ordinateur Amstrad CPC 464. Les différents chapitres traitent de l'animation — de caractères ou de dessins —, de la création de courbes et de diagrammes et de la possibilité d'utiliser des modèles de dessin.

Chaque chapitre comporte des programmes de démonstration qui présentent pour la plupart un intérêt réel. Par exemple, le Chapitre 2 contient un programme qui permet à l'utilisateur de concevoir ses propres caractères sur l'écran et de les sauvegarder dans un fichier ; le Chapitre 3 propose des sous-programmes qui autorisent la construction de diagrammes sectoriels ombrés et le Chapitre 4 un programme qui permet d'effectuer un dessin sur l'écran, d'en effacer n'importe quelle zone, de lui ajouter des détails et de sauvegarder l'image finale dans un fichier.

Ce livre suppose la connaissance du BASIC et de certaines notions telles que les boucles, les décisions et les sous-programmes. Bien qu'ils puissent utiliser de nombreux programmes tels quels, les débutants tireront certainement un meilleur profit de cet ouvrage en commençant par lire *Amstrad premiers programmes* de Rodnay Zaks (également publié chez Sybex).

Pour les lecteurs qui possèdent déjà une certaine pratique du BASIC mais qui n'ont pas lu ce premier livre, le Chapitre 1 en reprend quelques éléments qui permettent d'introduire les commandes graphiques élémentaires disponibles sur l'Amstrad.

1

NOTIONS ÉLÉMENTAIRES DE GRAPHISME

Il existe de nombreux types de supports papier destinés à des utilisations différentes. Un architecte ne dessine pas des plans de maisons sur un bloc-notes et un romancier ne se sert pas de copies d'écolier. En ce qui concerne le dessin sur ordinateur, l'écran est l'équivalent d'une feuille de papier et il est utile de pouvoir modifier son affichage en fonction du but poursuivi.

La commande MODE suivie du nombre 0, 1 ou 2 sélectionne l'un des trois modes d'affichage de l'Amstrad. Chaque mode autorise l'affichage sur l'écran d'un certain nombre de caractères par ligne, un nombre de couleurs simultanées particulier et un degré de résolution graphique défini (c'est-à-dire une certaine précision du dessin).

Mode	Nombre de lignes	Caractères par ligne
0	25	20
1	25	40
2	25	80

Figure 1.1 : Les trois modes d'affichage disponibles sur l'Amstrad CPC 464.

Lors de la frappe d'un texte important à partir du clavier, il est préférable de disposer d'une taille d'affichage maximale. Dans ce cas, le meilleur mode est le mode 2, car l'Amstrad peut alors afficher 25 lignes de 80 caractères.

L'ordinateur se replace automatiquement en mode 1 lors d'une réinitialisation ou de la mise hors tension. Ce mode comporte 25 lignes de 40 caractères. Le mode 1 est celui qui affiche les caractères les plus lisibles et peut être considéré comme le mode de "travail" c'est-à-dire celui que l'on utilise pour envoyer des commandes à l'ordinateur.

Le mode 0 affiche 25 lignes de 20 caractères ; c'est le plus intéressant pour la création de graphisme en couleur, car il permet l'emploi de seize couleurs simultanément.

Dans ces trois modes, le texte peut être affiché à n'importe quel endroit de l'écran, spécifié par ses coordonnées :

```
10 MODE 1
15 PRINT "Début d'affichage en (10,12)"
20 LOCATE 10,12
30 PRINT "C'est ici !"
```

La position (10,12) représente la position d'un caractère et XX,YY sont les *coordonnées texte* de cette position. Le premier nombre s'appelle coordonnée X et le second, coordonnée Y. Dans cet exemple, LOCATE 10,12 provoque l'affichage à partir de la dixième colonne et de la douzième ligne. On peut utiliser la commande CLS pour effacer l'écran après le déroulement du programme. La ligne 20 indique à l'Amstrad d'afficher ce qui suit aux coordonnées spécifiées dans la commande. Les coordonnées varient en fonction du mode, puisque chaque mode affiche un nombre de caractères par ligne différent.

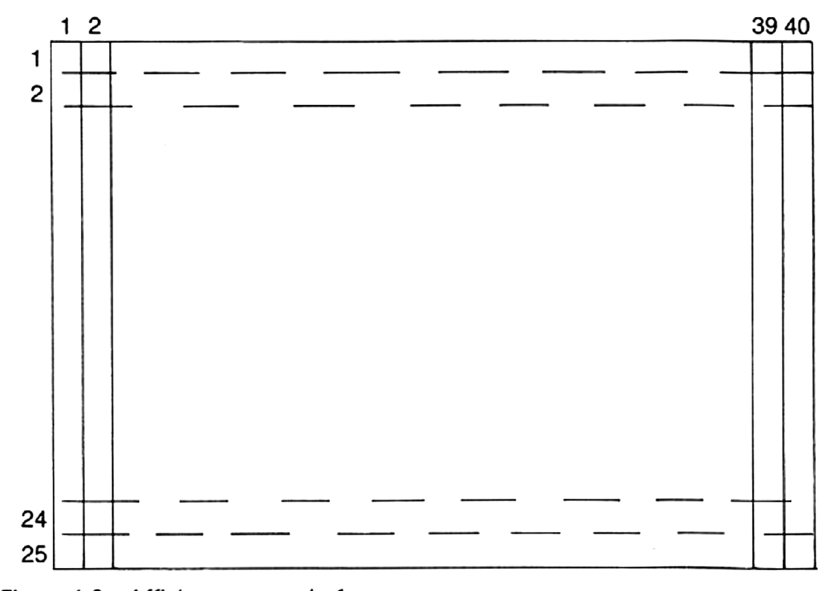


Figure 1.2 : Affichage en mode 1.

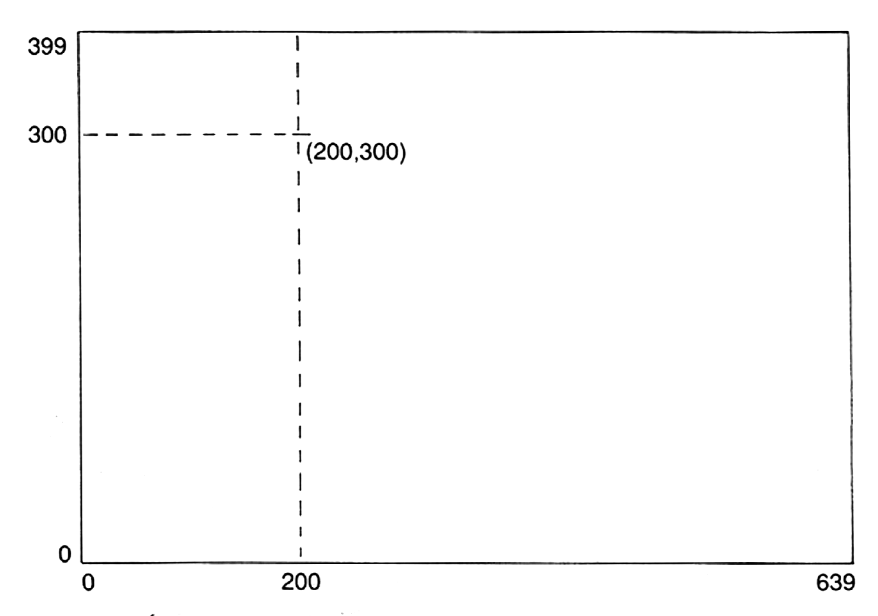


Figure 1.3 : Écran graphique indiquant le point (200,300).

En fait, l'affichage sur des positions de caractères n'offre que de tristes perspectives en matière de graphisme. Le mode 2 a 25 lignes de 80 caractères, mais l'élaboration d'un dessin en utilisant la position des caractères ne donne que de piètres résultats. Heureusement, chaque position peut être subdivisée en éléments plus petits appelés *pixels*. Cela permet l'affichage de lignes plus élégantes. Comme le nombre de caractères par ligne, la taille d'un pixel varie d'un mode à l'autre.

Cependant, ce système de coordonnées n'est pas adapté à la localisation des pixels, car chaque position de caractères se compose de plusieurs pixels. L'Amstrad utilise donc un système de coordonnées différent pour décrire la position d'un pixel et le localise à l'aide des *coordonnées graphiques*.

L'ÉCRAN GRAPHIQUE

Le système de coordonnées graphiques ressemble à celui des coordonnées texte, mais il ne fonctionne pas exactement de la même façon. L'écran graphique est divisé horizontalement en 640 points

et verticalement en 400 points. On peut identifier n'importe quelle position de cet écran.

La position du point de la Figure 1.3 est (200,300). On remarque que ces coordonnées graphiques sont évaluées à partir du bas de l'écran et que le point inférieur gauche a pour coordonnées (0,0). Cela peut prêter à confusion, car en coordonnées texte, le caractère supérieur gauche a pour coordonnées (1,1). On remarque également que, la numérotation commençant à 0, le point supérieur droit a pour coordonnées (639,399) et non pas (640,400).

Le programme suivant est une démonstration de deux commandes graphiques élémentaires disponibles sur l'Amstrad :

```
10 MODE 1
20 MOVE 124,156
30 DRAW 300,300
40 DRAW 200,400
50 DRAW 124,156
```

Nous utilisons ici le curseur graphique pour dessiner des lignes sur l'écran. Normalement, les curseurs graphique et texte sont confondus, mais, dès qu'une commande graphique est employée, c'est le curseur graphique jusqu'alors invisible qui est utilisé.

La commande MOVE de la ligne 20 provoque l'"apparition" du curseur graphique au point (124,156). La commande DRAW le déplace à partir de sa position (124,156) jusqu'aux nouvelles coordonnées (300,300) en traçant une ligne entre les deux points. Les deux commandes DRAW des lignes 40 et 50 dessinent les deux autres côtés du triangle.

En résumé, la commande MOVE x,y déplace le curseur graphique jusqu'au point x,y sans rien dessiner ; DRAW x,y dessine une ligne à partir du dernier point mentionné dans une commande MOVE ou DRAW et jusqu'au point de coordonnées x,y. Pour dessiner facilement des images complexes, il suffit de stocker les coordonnées x et y des points dans des instructions DATA puis de les lire.

```
10 MODE 0

20 x=1

29 REM dessin d' une figure avec MOVE et DRAW

30 WHILE x>0
```

```
40 READ x,y

50 READ x1,y1

60 MOVE x,y

70 DRAW x1,y1

80 LOCATE 1,24:PRINT x;" ";y;:r$="":WHILE r$="":
r$=INKEY$:WEND

90 WEND

100 END

110 DATA 308,162,344,174,344,174,360,216,360,216
,364,260,364,260,360,310

120 DATA 360,310,336,344,336,344,292,356,292,356
,248,352,248,352,212,342

130 DATA 212,342,200,316,200,316,196,246,196,246
,200,214,200,214,216,174

140 DATA 216,174,252,166,316,198,300,190,300,190
,272,190

150 DATA 272,190,256,198,244,200,244,210,244,204
,324,204,324,208,324,198

160 DATA 256,238,268,232,268,232,288,232,288,232
,296,242,276,242,276,284

170 DATA 288,284,300,296,300,296,324,286,324,286
,288,282,264,282,252,294

180 DATA 252,294,232,280,232,280,264,280,196,280
,180,296,180,296,172,266

190 DATA 172,266,192,236,364,236,384,266,384,266
,380,294,380,294,364,282

200 DATA 320,348,332,324,312,324,312,352,288,354
,292,326,272,354,280,326,264,348

210 DATA 224,344,216,334,264,344,268,228,268,218
,272,216,272,230,276,228,276,214,280,214,280,230
,280,230,284,212

220 DATA 288,212,284,230,288,228,288,220

230 DATA -1,-1
```

Des effets impressionnants peuvent être générés à l'aide des deux instructions MOVE et DRAW. On peut élaborer des courbes à partir de lignes droites en déplaçant les extrémités de ces lignes d'une distance déterminée à chaque itération :

```
1 MODE 1
10 x=100: y=100
20 maximum=300
30 stepsize=10
40 FOR number=0 TO maximum STEP stepsize
50 MOVE x+number, y
60 DRAW x+maximum, y+number
70 MOVE x, y+number
80 DRAW x+number, y+maximum
90 NEXT
```

LA RÉSOLUTION DES DIFFÉRENTS MODES

Bien que l'écran graphique soit divisé en 640 points horizontalement et 400 points verticalement, l'Amstrad ne peut pas réellement les individualiser. L'écran graphique est le même dans tous les modes, mais la distinction entre les points est meilleure dans l'un des modes que dans les autres. Nous allons à nouveau faire tourner le programme après avoir modifié la ligne 1 :

```
1 MODE 0
```

Le dessin reste inchangé, mais les traits qui le composent sont plus épais et l'image semble plus "grossière". On tape maintenant :

```
10 MODE 2
20 MOVE 200, 300
30 DRAW 200, 399
```

```
40 MOVE 201,200
50 DRAW 201,399
60 MOVE 202,100
70 DRAW 202,399
80 MOVE 203,0
90 DRAW 203,399
```

Cette fois-ci, les lignes sont parfaites. Le mode 2 est appelé mode *haute résolution* car il permet à l'Amstrad de distinguer 640 points horizontaux et 200 points verticaux. Grâce à cela, la commande DRAW génère de très jolies lignes.

En mode 2, l'Amstrad ne fait pas de différence entre deux points dont les coordonnées verticales sont trop proches. Il traite les points (10,10) et (10,11) comme étant parfaitement identiques. En fait, les modes 1 et 0 ont la même résolution verticale de 200 points que le mode 2, mais leur résolution horizontale est beaucoup moins bonne. On tape :

```
10 MODE 1
20 MOVE 200,300
30 DRAW 200,399
40 MOVE 201,200
50 DRAW 201,399
60 MOVE 202,100
70 DRAW 202,399
80 MOVE 203,0
90 DRAW 203,399
```

puis on exécute à nouveau le programme. Le mode 1 est le mode *résolution intermédiaire* ; dans ce mode, l'Amstrad ne distingue que 320 points horizontaux. Cela signifie par exemple que (200,300) et (201,300) sont traités comme des points identiques. On tape ensuite :

```
10 MODE 0
```

puis on fait tourner le programme pour la troisième fois. Le mode 0 est le mode *basse résolution* et ne peut identifier que 160 points horizontaux différents.

Il y a bien sûr une autre raison qui justifie l'utilisation d'un mode générant des images grossières alors que le mode 2 est accessible. En effet, bien que le mode 0 soit le mode basse résolution, il permet l'affichage de dessins en seize couleurs simultanées. On voit, Figure 1.4, que les modes 1 et 2 n'offrent pas les mêmes possibilités.

Mode	Résolution graphique	Nombre de couleurs affichées simultanément
0	160200	2
1	320200	4
2	640200	16

Figure 1.4 : Les différentes résolutions graphiques et le nombre de couleurs disponibles dans chacun des modes.

L'Amstrad a une taille mémoire limitée. Il ne peut enregistrer dans sa RAM (mémoire vive) qu'une certaine quantité d'informations relatives à l'écran. Comme c'est souvent le cas en informatique, il faut faire un choix ; la RAM peut servir à enregistrer de très nombreux points avec deux couleurs possibles, ou un nombre de points moins important avec quatre couleurs possibles, ou encore bien moins de points avec un choix de seize couleurs. L'ordinateur offre plusieurs possibilités et il faut sélectionner le mode qui semble le mieux correspondre au but recherché.

L'INSTRUCTION PLOT

Chacune des lignes tracées dans les programmes précédents était effectivement constituée d'un certain nombre de pixels. L'Amstrad peut afficher des pixels individuellement, bien qu'un pixel unique soit relativement difficile à visualiser en mode 2. En fait, chaque pixel est constitué d'un certain nombre de points que la précision des trois modes ne permet pas d'identifier. Dans chacun de ces modes, le pixel est le plus petit groupe de points qui puisse être localisé.

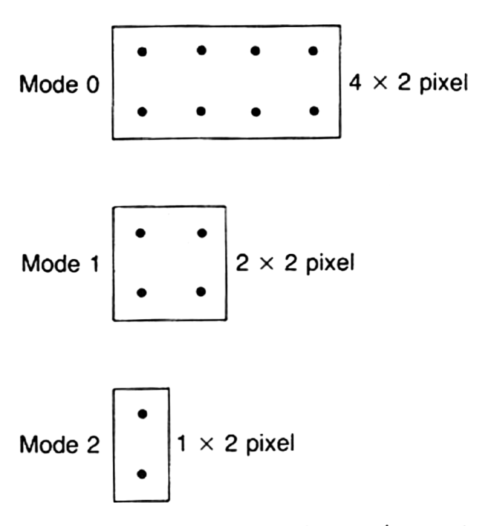


Figure 1.5 : Taille d'un pixel graphique dans chacun des modes.

Il peut paraître étrange de disposer de plus de points qu'il n'est possible d'en afficher, mais cela permettra des améliorations ultérieures de la résolution graphique sans que l'on ait à modifier complètement le système de coordonnées.

L'instruction PLOT s'emploie comme les instructions MOVE et DRAW. Elle doit être suivie des coordonnées x et y du pixel à tracer. Le programme suivant dessine six pixels sur l'écran.

```
10 MODE 0
20 PLOT 160,200
30 PLOT 320,200
40 PLOT 324,200
50 PLOT 328,200
60 PLOT 332,200
70 PLOT 480,200
```

En mode 0, la résolution est si basse que les quatre pixels tracés lignes 30 à 60 se rejoignent pour former une ligne ; en mode 1, les pixels sont individualisés et en mode 2, ils sont si petits qu'on ne les distingue plus.

A la mise sous tension, l'ordinateur affiche, dans tous les modes, des caractères et des dessins jaunes sur fond bleu. Il existe en fait vingt-sept couleurs différentes, même si certaines d'entre elles sont difficiles à distinguer. Chaque couleur possède un numéro associé à l'instruction INK, que l'on utilise à la place du nom de la couleur chaque fois que l'on veut s'y référer.

INK nombre	Couleur
0	Noir
1	Bleu
2	Bleu vif
3	Rouge
4	Magenta
5	Mauve
6	Rouge vif
7	Pourpre
8	Magenta vif
9	Vert
10	Turquoise
11	Bleu ciel
12	Jaune
13	Blanc
14	Bleu pastel
15	Orange
16	Rose
17	Magenta pastel
18	Vert vif
19	Vert marin
20	Turquoise vif
21	Vert citron
22	Vert pastel
23	Turquoise pastel
24	Jaune vif
25	Jaune pastel
26	Blanc brillant

Figure 1.6 : Les vingt-sept couleurs d'encre (INK) disponibles sur l'Amstrad CPC 464.

A ce stade, il est important de savoir que l'ordinateur n'utilise pas vraiment la totalité de l'écran pour son affichage. Il opère à l'intérieur d'un vaste rectangle entouré d'une bordure qui n'est pas utilisée. Bien qu'elle ne soit pas employée, cette bordure a la même couleur que le reste de l'écran, mais elle ne fait pas réellement partie de la mémoire de l'ordinateur, puisqu'elle ne sert jamais à l'affichage de caractères ou de dessins.

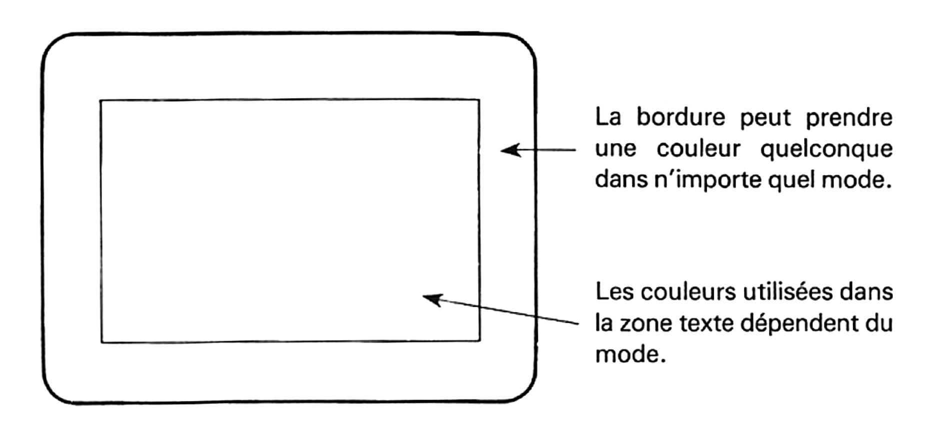


Figure 1.7 : Bordure de l'écran sur un moniteur ou un récepteur de télévision.

On peut attribuer à la bordure *n'importe quelle couleur* et *n'importe quel mode*. Le choix de sa couleur ne subit aucune restriction ; le mode 2 ne permet d'afficher que deux couleurs simultanément à l'intérieur du rectangle, mais la couleur de la bordure est une couleur quelconque. On tape :

```
10 MODE 2
20 BORDER 0
```

On voit, Figure 1.6, que 0 est le nombre correspondant à la couleur noire. La frappe de BORDER 0 génère l'affichage d'une bordure noire. On peut s'entraîner à établir d'autres couleurs, n'importe quel numéro compris entre 0 et 26 pouvant être employé (c'est-à-dire qu'il existe 27 couleurs possibles). Par exemple, BORDER 26 génère une bordure blanche.

La couleur de la bordure peut être établie exactement de la même manière en mode 0 ou en mode 1. Quand on change de mode après la définition de la couleur de la bordure, celle-ci garde sa couleur. A la mise sous tension ou après une initialisation, la bordure est bleue, c'est-à-dire BORDER 1.

LES COULEURS DU "STYLO" ET DU "PAPIER"

Les couleurs utilisées dans le rectangle principal peuvent également être modifiées. Le problème de la mémoire RAM devient alors crucial et le nombre de couleurs qui peuvent être affichées simultanément sur l'écran subit quelques restrictions.

Il est possible de modifier la couleur du "stylo" de l'Amstrad à l'aide de la commande PEN. On tape :

```
MODE 0
PEN 4
```

D'après la Figure 1.6, cette commande devrait donner la couleur magenta aux caractères, mais la sélection des couleurs de l'écran principal n'est pas identique à celle de la bordure. Le choix de PEN 4 entraîne un affichage de caractères blancs et celle de PEN 5 celui de caractères noirs.

On peut même obtenir des caractères clignotants bleu/jaune en tapant :

```
PEN 14
```

Il y a seize possibilités utilisables dans n'importe quel mode et la Figure 1.8 montre les numéros des stylos et les couleurs d'encre correspondantes dans chaque mode. On remarque que ce numéro peut correspondre à des couleurs différentes selon le mode. Cela signifie qu'un programme fonctionnant parfaitement en mode 0 peut générer un écran vide en mode 2 si le numéro choisi pour la commande PEN correspond à la couleur du fond dans ce mode. Nous voyons que les seize possibilités ne sont pas d'une grande utilité en mode 2, car huit d'entre elles correspondent au jaune et les huit autres au bleu. Nous verrons plus loin comment modifier les couleurs qui peuvent être employées dans chaque mode. La couleur du fond peut égale-

ment être modifiée à l'aide de la commande PAPER. On initialise l'ordinateur en maintenant enfoncées les touches CTRL et SHIFT et en tapant la touche ESC ; l'écran est en mode 0, on tape :

PAPER 3

Les caractères tapés à partir de maintenant sont affichés sur fond rouge. L'ensemble du rectangle peut avoir cette nouvelle couleur si on utilise la commande CLS.

Numéro PEN ou PAPER	Mode 0	Mode 1	Mode 2
0	1	1	1
1	24	24	24
2	20	20	1
3	6	6	24
4	26	1	1
5	0	24	24
6	2	20	1
7	8	6	24
8	10	1	1
9	12	24	24
10	14	20	1
11	16	6	24
12	18	1	1
13	22	24	24
14	1/24	20	1
15	16/11	6	24

Figure 1.8 : Couleurs du stylo (PEN) ou du papier (PAPER) dans les différents modes. En mode 0, la commande PEN ou PAPER 14 ou 15 génère un affichage clignotant dans les deux couleurs indiquées.

En mode 0, PAPER a les seize mêmes couleurs que PEN. PEN 14 génère un affichage de caractères clignotants bleu/jaune et PAPER 14 l'affichage d'un fond clignotant bleu/jaune. La Figure 1.6 peut aider l'utilisateur à sélectionner ses couleurs. Par exemple, pour obtenir des caractères rouges sur fond blanc en mode 0, on tape :

```
INK 3
PAPER 4
CLS
```

Bien sûr, les commandes PEN et PAPER peuvent être insérées dans des programmes.

```
10 MODE 0
20 LOCATE 4,7
30 PEN 3
40 PAPER 5
50 PRINT"rouge sur noir"
60 LOCATE 4,13
70 PEN 6
80 PAPER 3
90 PRINT"bleu sur rouge"
100 LOCATE 4,19
110 PEN 5
120 PAPER 6
130  PRINT"noir sur bleu"
140 REM retour a la normale
150 PEN 1
160 PAPER 0
```

On modifie la ligne 10, puis on fait tourner ce programme dans les deux autres modes. On obtient alors des résultats amusants car les couleurs sont différentes dans les autres modes. Voici un autre exemple :

```
10 MODE 0
20 redpeninmode0=3
30 blackpaperinmode0=5
40 PEN redpeninmode0
50 PAPER blackpaperinmode0
```

```
60 CLS
70 LOCATE 8,12
80 PRINT"ca y est"
90 REM retour a la normale
100 PEN 1
110 PAPER 0
```

Les deux dernières lignes restituent les couleurs initiales.

Un problème se pose fréquemment quand on joue avec les couleurs : si la couleur d'écriture est identique à celle du fond, toute lecture est impossible. On peut contourner cette difficulté en restituant la couleur initiale avant la fin du programme. Une autre solution consiste à programmer l'une des touches fonction de l'Amstrad pour qu'elle initialise les couleurs d'écriture et de fond :

```
KEY 128, CHR$(13) + "INK 0,1 : INK 1,24" + CHR$(13)
```

EXERCICES

1. Écrire un programme qui détermine un point aléatoire sur l'écran et le relie par un trait à un autre point aléatoire dans une couleur également aléatoire. Le processus reprend à partir du deuxième point et se répète jusqu'à ce que 100 lignes aient été tracées.
2. Tracer une fusée, à l'aide des commandes MOVE et DRAW. Donner à cette fusée un nom qui sera affiché.
3. Afficher "Différentes couleurs" au milieu de l'écran en mode 0, chaque lettre ayant une couleur différente.

GRAPHISME ET COULEUR

L'élaboration de dessins à partir de traits colorés est facile à réaliser sur l'Amstrad. Les commandes MOVE et DRAW employées seules tracent toujours des lignes de couleur PEN 1 quel que soit le mode.

Tous les programmes graphiques que nous avons vus jusqu'à présent ont produit des tracés de couleur PEN 1. PEN 1 correspond à l'encre (INK) 24 dans tous les modes et les lignes sont donc toutes jaune vif.

Pour obtenir une couleur différente, il faut ajouter une extension à la commande DRAW. On réinitialise l'Amstrad et on tape :

```
MOVE 100,100
DRAW 300,300,2
```

Une ligne rouge est dessinée avec le stylo PEN 3 entre les points (300,300) et (400,0). En mode 1, PEN 3 utilise l'encre n° 6.

Ces commandes sont aussi faciles à insérer dans un programme. Il ne faut pas oublier qu'un programme qui tourne dans un mode n'est pas toujours exécutable dans un autre mode à cause des modifications de numéros d'encre (INK) entre les différents modes. En mode 1, le programme suivant dessine un rectangle comportant un côté jaune, un cyan et deux côtés rouges.

```
10 MODE 1
20 MOVE 100,100
30 DRAW 400,100
40 DRAW 400,300,3
50 DRAW 100,300,2
60 DRAW 100,100,3
```

On remarque que, dans la ligne 30, aucun numéro n'est spécifié pour la commande PEN ; dans ce cas, l'Amstrad utilise automatiquement PEN 1, qui trace une ligne jaune. Si on exécute à nouveau le programme, on s'aperçoit qu'il n'y a plus de ligne jaune.

Chaque fois que l'ordinateur rencontre une commande graphique comme DRAW sans que le numéro de PEN soit spécifié, il emploie le numéro en cours pour exécuter la commande. Au premier déroulement, l'Amstrad utilise PEN 1, ligne 30. Après exécution, le dernier PEN employé est PEN 3. Cette couleur devient donc la couleur en cours et, lors du second déroulement, PEN 3 est employé dans la ligne 30, puisqu'aucune valeur n'a été spécifiée pour le stylo. Après l'attribution d'une couleur de stylo dans une commande DRAW,

toutes les lignes suivantes sont tracées de cette couleur, sauf si une nouvelle valeur est introduite pour PEN :

```
10 MODE 1
20 MOVE 200,100
30 DRAW 400,200,2
40 DRAW 100,350
50 DRAW 200,100
```

On peut faire tourner ce programme en mode 2, dans lequel PEN 2 a une couleur d'encre différente.

Le mode 0 est de loin le meilleur mode pour générer un affichage graphique couleur dans la mesure où l'on ne s'attache pas trop à la résolution.

```
10 MODE 0
20 x = 0 : y = 0
30 couleur = 1
40 FOR compt = 0 TO 350 STEP 4
50 MOVE x+compt,y
60 DRAW x+350 ,y+compt,couleur
70 MOVE x,y+compt
80 DRAW x+compt,y+350
90 couleur = (couleur+1) MOD 16
```

MODIFICATION DE COULEUR

Jusqu'à présent, nous n'avons vu que quelques unes des couleurs que l'Amstrad peut produire. Il n'y a que seize valeurs de stylo (PEN) disponibles, et pourtant la Figure 1.6 montre que l'on peut utiliser vingt-sept valeurs d'encre (INK). L'Amstrad permet de modifier la valeur d'encre de chaque stylo de telle sorte qu'il est possible de choisir n'importe quelle combinaison de couleurs pour un mode particulier.

Cependant, le nombre de couleurs qui peuvent être employées simultanément sur l'écran ne change pas. En mode 2, seules deux couleurs sont autorisées ; en mode 1, quatre couleurs et en mode 0, seize couleurs. Lors de la mise sous tension ou de l'initialisation de l'ordinateur, celui-ci se place automatiquement en mode 1 et utilise

PAPER 0 qui est bleu (numéro d'encre 1) dans tous les modes et PEN 1 qui est jaune (numéro d'encre 24) également dans tous les modes. On réinitialise l'ordinateur puis on tape :

```
INK 1,6
```

Tout le texte affiché en jaune sur l'écran devient instantanément rouge vif. La commande INK nécessite deux nombres. Le premier est le nombre associé à PEN ou PAPER dont la couleur doit être modifiée. Le second représente la couleur d'encre que l'on veut attribuer.

La commande INK 1,6 indique à l'ordinateur d'attribuer à PEN 1 la couleur d'encre (INK) 6, c'est-à-dire rouge vif. Tout ce qui a été tapé ou dessiné précédemment devient rouge.

Pour remplacer la couleur en cours par la couleur bleu, on tape

```
INK 1,2
```

Ce qui était rouge devient bleu. Pour revenir à la couleur normale, on tape :

```
INK 1,24
```

On voit, en regardant la Figure 1.8, que PEN 1 est associé à INK 24 dans tous les modes.

Il est aussi facile de modifier la couleur du papier (PAPER). Pour l'instant, l'Amstrad utilise PAPER 0, c'est-à-dire le bleu. Nous allons attribuer à l'affichage la couleur blanc :

```
INK 0,26
```

Si le texte est difficile à lire, on peut essayer :

```
INK 0,6
```

ou alors :

```
INK 0,0
```

Dans tous les modes, PAPER est normalement bleu, avec pour couleur d'encre (INK) 1.

Il n'est pas nécessaire d'avoir déjà utilisé la commande PEN ou PAPER pour la modifier. On réinitialise l'ordinateur, puis on tape :

INK 3,0

Apparemment, il ne se passe rien. D'après la Figure 1.8, si on choisit PEN 3 en mode 1, le texte sera affiché avec la couleur 6, c'est-à-dire rouge vif. Mais nous venons d'employer la commande INK pour attribuer à PEN 3 la couleur INK 0, c'est-à-dire noir. On tape :

PEN 3

Tout le texte apparaît en noir. On tape ensuite :

INK 3,6

Le stylo (PEN 3) et tout ce qu'il affiche maintenant correspond au numéro de couleur 6, c'est-à-dire rouge vif. Cette modification de couleur reste valide même si on change de mode.

Il est également possible d'établir deux couleurs clignotantes :

INK 1,3,26

Le texte affiché par PEN 1 clignote dans les deux couleurs, rouge (INK 3) et blanc (INK 26).

Une sélection de couleurs clignotantes bien choisies peut servir, dans un programme, à donner l'illusion du mouvement. On peut par exemple attribuer à PEN 1 un clignotement jaune/rouge et à PEN 2 un clignotement rouge/jaune. En affichant des caractères alternés avec des stylos alternés, on peut créer un effet d'''ondulation'' donnant l'illusion du déplacement de la couleur sur la ligne :

```
10 MODE 1
20 INK 1,3,12
30 INK 2,12,13
40 pencolour=1
50 FOR x=1 TO 40
```

```
60 IF pencolour=1 THEN pencolour=2 ELSE pencolou
r=1

70 PEN pencolour

80 LOCATE x,13

90 PRINT CHR$(143);

100 NEXT
```

La commande INK présente un avantage considérable ; elle permet de choisir n'importe quelle combinaison de couleurs parmi les vingt-sept disponibles. Même dans le mode 2 (qui n'autorise que deux couleurs), on peut mettre un affichage en valeur en plaçant un texte rouge sur un fond blanc au lieu de se limiter aux seules couleurs jaune et bleue disponibles à la mise sous tension. Le programme suivant montre les 700 combinaisons de couleurs disponibles en mode 2 :

```
10 MODE 2

20 FOR x=0 TO 27

30 CLS

40 INK 0,x

50 FOR y=0 TO 27

60 IF x<>y THEN INK 1,y:PRINT"Encre ";y

70 reponse$=""

80 WHILE reponse$=""

90 reponse$=INKEY$

100 WEND

110 NEXT

120 NEXT
```

EXERCICES

1. Afficher un nom en caractères clignotants sur l'écran. Choisir la couleur du fond de telle sorte que les lettres semblent apparaître et disparaître.
2. Dessiner un feu en employant les couleurs appropriées pour les lignes (on peut employer des couleurs clignotantes si nécessaire).
3. Dessiner un crabe rouge sur une plage de sable jaune. Utiliser des paramètres pour les instructions PEN et INK de telle sorte que les couleurs soient les mêmes quel que soit le mode choisi lors du déroulement du programme.

2

CODES ET CARACTÈRES

LE JEU DE CARACTÈRES DE L'AMSTRAD

L'Amstrad peut afficher sur l'écran de nombreux caractères. En plus des caractères alphabétiques et alphanumériques habituels, il peut générer des caractères graphiques comme on le voit dans ce programme :

```
10 MODE 1
20 FOR code = 32 TO 255
30 PRINT CHR$(code);
40 NEXT
```

L'Amstrad associe chaque caractère à un numéro de code appelé code ASCII ; ce numéro est compris entre 0 et 255. Les codes 0 à 31 ont une signification particulière, comme par exemple *Reculer le curseur d'un espace* ou *Modifier la couleur d'encre INK*. Les codes 32 à 255 servent pour l'alphabet minuscule et majuscule, les chiffres, la ponctuation, etc. La ligne 30 du programme indique à l'ordinateur *Afficher le caractère qui a le code ASCII suivant*. La Figure 2.1 montre les codes ASCII les plus utiles.

Caractère	Codes ASCII
<	0-31
espace	32
0-9	48-57
A-Z	65-90
a-z	97-122

Figure 2.1 : Les codes ASCII les plus utiles.

Les caractères dont le code est compris entre 0 et 31 peuvent être affichés s'ils sont précédés du caractère dont le code ASCII est 1 :

```
10 MODE 1
20 FOR code = 0 TO 31
30 PRINT CHR$(1) CHR$(code);
40 NEXT
```

Cela laisse de grandes possibilités dans le choix des caractères à utiliser dans un programme. Cependant, il arrive souvent que le jeu de caractères de l'Amstrad ne contienne pas le caractère désiré, par exemple lors de l'utilisation d'une langue étrangère ou d'un travail mathématique, ou encore dans un programme de jeu. On utilise alors la possibilité offerte par l'ordinateur de générer des caractères *définis par l'utilisateur*. Nous allons d'abord voir comment l'Amstrad stocke les caractères et pourquoi cette méthode de stockage nous limite à 256 caractères prédéfinis avec les codes ASCII 0 à 255.

BITS, OCTETS ET SYSTÈME BINAIRE

Comment l'Amstrad stocke-t-il les informations ? Pour simplifier, on imagine que l'ordinateur contient des milliers de commutateurs, chacun d'entre eux pouvant être ''ouvert'' ou ''fermé''. Dans l'Amstrad, ces commutateurs sont groupés par huit. Si on représente un commutateur fermé par 0 et un commutateur ouvert par 1, toutes les combinaisons possibles sont celles qui apparaissent Figure 2.2.

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 1
      .
      .
      .
1 1 1 1 1 1 0 1
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1
```

Figure 2.2 : Les 256 combinaisons possibles des 8 ''commutateurs''.

Il n'est pas très étonnant de constater qu'il existe 256 combinaisons possibles. Chacun de ces nombres est un *nombre binaire* uniquement composé des chiffres 0 et 1 appelés *bits*. Chaque combinaison de huit bits s'appelle un *octet*.

Le système binaire est un système de calcul en base deux, comme on compte en base dix, et chaque nombre binaire équivaut à un nombre de notre système de calcul. Il est relativement facile de

convertir n'importe quel nombre binaire en nombre décimal, comme nous allons le voir un peu plus loin.

L'octet est l'unité de base du stockage de l'information sur l'Amstrad. De nombreuses restrictions de la machine viennent du fait qu'un octet (de huit bits) peut être "établi" de 256 manières différentes.

Il n'y a que 256 caractères prédéfinis parce que chaque caractère correspond à un code de référence ASCII qui est stocké dans un seul octet. S'il existait 257 caractères prédéfinis, les informations seraient stockées sur deux octets et cela entraînerait une consommation d'espace mémoire beaucoup plus importante.
A chaque caractère prédéfini est associé un code ASCII.

Chaque caractère est élaboré sur une grille de huit cases sur huit, comme le montre la Figure 2.3.

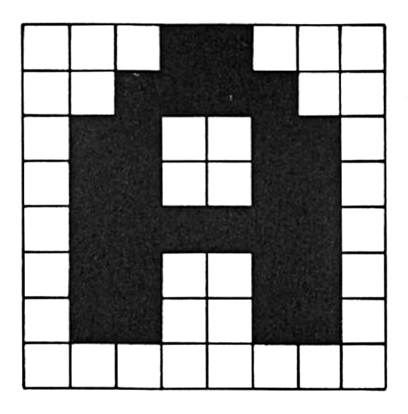


Figure 2.3 : Lettre majuscule A.

Chaque case de la grille est ouverte ou fermée (allumée ou éteinte)... ce qui fait penser aux nombres binaires. En effet, la grille de huit cases sur huit est une autre conséquence de la présence des octets de huit bits. Chaque rangée de la grille peut être stockée comme un octet et, par conséquent, la description complète d'un caractère peut être stockée dans huit octets.

```
0 0 0 1 1 1 0 0
0 0 1 1 1 1 0 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 1 1 1 1 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 0 0 0 0 0 0 0
```

Figure 2.4 : Définition des huit octets de la lettre A.

Il n'est pas très facile de travailler sur les nombres binaires, car ils sont encombrants et il est fréquent d'introduire ou d'oublier un 0 ou un 1 par erreur. On peut convertir les valeurs binaires des octets en valeurs décimales en faisant la somme des valeurs pondérées de chaque bit pour chacune des rangées. L'Amstrad peut simplifier considérablement ce processus en effectuant ce travail à notre place ; la commande PRINT STR$(nombre) convertit un nombre en une chaîne décimale équivalente. Les nombres binaires doivent commencer par "&X", sinon l'Amstrad considère le nombre comme un nombre décimal très grand constitué de 0 et de 1. Nous allons vérifier que nos calculs sont exacts :

```
1
2 6 3 1
8 4 2 6 8 4 2 1
0 0 0 1 1 0 0 0                        168     24
0 0 1 1 1 1 0 0                     321684     60
0 1 1 0 0 1 1 0                     643242    102
0 1 1 0 0 1 1 0                     643242    102
0 1 1 1 1 1 1 0                  643216842    126
0 1 1 0 0 1 1 0                     643242    102
0 1 1 0 0 1 1 0                     643242    102
0 0 0 0 0 0 0 0                                 0
```

Figure 2.5 : Définition de la lettre A à l'aide des nombres décimaux.

```
10 MODE 1
20 nombre = 1
30 WHILE nombre > 0
40 INPUT "Entrer le nombre binaire
precede de &X.",nombre
50 PRINT "Voici la valeur décimale de ce
nombre" STR$(nombre)
60 WEND
```

On a parfois besoin de convertir des nombres décimaux en nombres binaires, ce que l'Amstrad fait également :

```
10 MODE 1
20 nombre = 1
30 WHILE nombre > 0
40 INPUT "Entrer le nombre decimal",nomb
re
50 PRINT "Voici la valeur binaire de ce
nombre" BIN$(nombre)
60 WEND
```

On remarque, dans ces deux conversions, que le résultat est une *chaîne*. Comme il est impossible d'effectuer des opérations arithmétiques sur une chaîne, il faut d'abord la convertir en un nombre :

```
10 MODE 1
20 number=1
30 WHILE number>0
40 INPUT"taper un nombre decimale ",number
50 PRINT"voici le nombre binaire "BIN$(number)
55 numeric=VAL(BIN$(number))
56 PRINT"voici le nombre ",numeric
60 WEND
```

DÉFINITION DE CARACTÈRES PROPRES A L'UTILISATEUR

Nous connaissons maintenant les valeurs des huit octets que l'Amstrad utilise pour décrire la lettre A. Il est possible de redéfinir n'importe lequel des seize caractères dont le code ASCII est compris entre 240 et 255 ; nous allons donc remplacer le caractère 240 par la lettre A :

```
10 MODE 1
20 REM SYMBOL definit un caractere
30 SYMBOL 240,24,60,102,102,126,102,102,
0
40 PRINT CHR$(240)
```

L'instruction SYMBOL, ligne 30, indique à l'Amstrad que nous voulons définir un nouveau caractère. Le premier nombre, 240, donne le code ASCII du caractère et les huit nombres suivants définissent chaque "rangée" de la grille du caractère.

Pour définir plus de seize codes ASCII, il faut employer l'instruction SYMBOL AFTER :

```
10 MODE 1
20 SYMBOL AFTER 65
30 SYMBOL 65,231,195,153,153,129,153,153,
255
40 PRINT CHR$(65)
```

La ligne 20 indique à l'ordinateur que nous voulons redéfinir les codes ASCII supérieurs ou égaux à 65. La ligne 30 redéfinit la lettre majuscule A pour que les points allumés s'éteignent et vice versa ; ce caractère a le code ASCII 65. La définition précédente est perdue ; on peut l'obtenir à nouveau en réinitialisant la machine ou en utilisant une autre instruction SYMBOL AFTER qui réinitialise tous les caractères à leur définition initiale :

```
10 MODE 1
20 SYMBOL AFTER 65
30 SYMBOL 65,231,195,153,153,129,153,153,
255
40 PRINT CHR$(65)
50 SYMBOL AFTER 70
```

SYSTÈME HEXADÉCIMAL

Nous avons vu que l'Amstrad peut facilement convertir des nombres binaires en nombres décimaux. Mais il est d'usage en informatique d'utiliser le système *hexadécimal*, c'est-à-dire de calculer en base seize.

Nombre décimal	Équivalent hexadécimal
0	0
1	1
.	.
.	.
10	A
11	B
12	C
13	D
14	E
15	F
16	10
17	11
.	.
30	1E
.	.
100	64
.	.
255	FF

Figure 2.6 : Quelques nombres décimaux et leurs équivalents hexadécimaux.

Les nombres hexadécimaux sont généralement précédés du signe "&" pour éviter toute confusion avec les nombres décimaux. Il est conseillé de se familiariser avec les nombres hexadécimaux ; nous allons donc définir la lettre A dans le système hexadécimal :

```
10 MODE 1
20 SYMBOL 240,&18,&30,&66,&66,&7E,&66,&6
6,0
30 PRINT CHR$(240)
```

Étant donné que l'Amstrad travaille avec des octets, des nombres décimaux sans signification apparente prennent leur signification en hexadécimal. On s'aperçoit par exemple que l'Amstrad rejette tout numéro de ligne supérieur à 65535. Ce nombre semble tout à fait arbitraire, mais, une fois converti en hexadécimal, il a pour valeur &FFFF et la raison du rejet devient claire : 65535 est le plus grand nombre que peuvent contenir deux octets ; l'emploi de deux octets signifie qu'il y a 256 × 256 = 65535 différents numéros de ligne disponibles.

L'Amstrad permet d'effectuer facilement la conversion des nombres décimaux en nombres hexadécimaux :

```
10 MODE 1
20 nombre=1
30 WHILE nombre>0
40 INPUT "Entrer le nombre decimal",nom
bre
50 PRINT "En hexadecimal ce nombre est"
,HEX$(nombre)
60 WEND
```

La conversion du système hexadécimal en système décimal nécessite à nouveau l'emploi de l'instruction PRINT STR$(nombre), mais cette fois-ci, pour indiquer que le nombre concerné est un nombre hexadécimal, il est précédé du signe "&" :

```
10 MODE 1
20 nombre=1
30 WHILE nombre>0
40 INPUT "Entrer le nombre hexadecimal p
recede de &", nombre
50 PRINT "En decimal ce nombre est",STR
$(nombre)
60 WEND
```

JEUX

Les caractères définis par l'utilisateur sont particulièrement utiles dans les programmes de jeux. Le programme suivant définit un carac-

tère "chien" qui est ensuite affiché à divers endroits de l'écran, de manière aléatoire :

```
10 MODE 0
20 SYMBOL 240,0,4,7,132,124,130,130,0
30 FOR randomdogs=1 TO 30
40 randomx=INT(19*RND(1)+1)
50 randomy=INT(24*RND(1)+1)
60 pencolour=INT(15*RND(1)+1)
70 PEN pencolour
80 LOCATE randomx,randomy
90 PRINT CHR$(240)
100 NEXT
```

Pour déplacer le chien sur l'écran, il faut afficher un espace sur la position en cours (pour effacer l'ancien caractère), puis afficher la nouvelle position :

```
10 MODE 0
20 SYMBOL 240,0,4,7,132,124,130,130,0
30 dog$=CHR$(240)
40 PEN 1
50 dogx=13: dogy=10
60 reponse$=""
70 WHILE reponse$<>"e"
80 newy=dogy: newx=dogx
90 reponse$=INKEY$
100 IF reponse$="a" AND dogy>1 THEN newy=dogy-1
110 IF reponse$="z" AND dogy<25 THEN newy=dogy+1
```

```
120 IF reponse$="," AND dogx>1 THEN newx=dogx-1

130 IF reponse$="." AND dogx<20 THEN newx=dogx+1

140 IF dogx<>newx OR dogy<>newy THEN LOCATE dogx
,dogy:PRINT" ";:dogx=newx:dogy=newy

150 LOCATE dogx,dogy

160 PRINT dog$

170 WEND
```

On peut rendre l'animation plus intéressante en définissant une série de caractères qui ne sont affichés que si le déplacement est effectué dans une direction particulière. On peut modifier le programme ci-dessus pour illustrer cette possibilité ; au lieu de définir quatre nouveaux caractères, nous allons nous servir des caractères prédéfinis dont les codes ASCII sont compris entre 240 et 243. Ces caractères sont des flèches pointant dans différentes directions.

```
10 MODE 0

20 arrow$=CHR$(240)

40 PEN 1

50 arrowx=13:arrowy=10

60 reponse$=""

70 WHILE reponse$<>"e"

80 newy=arrowy:newx=arrowx

90 reponse$=INKEY$

100 IF reponse$="a" AND arrowy>1 THEN newy=arrow
y-1:arrow$=CHR$(240)

110 IF reponse$="z" AND arrowy<25 THEN newy=arro
wy+1:arrow$=CHR$(241)

120 IF reponse$="," AND arrowx>1 THEN newx=arrow
x-1:arrow$=CHR$(242)

130 IF reponse$="." AND arrowx<20 THEN newx=arro
wx+1:arrow$=CHR$(243)
```

```
140 IF arrowx<>newx OR arrowy<>newy THEN LOCATE
arrowx, arrowy: PRINT" ";: arrowx=newx: arrowy=newy

150 LOCATE arrowx, arrowy

160 PRINT arrow$

170 WEND
```

La structure de ce programme peut servir de base à de nombreux jeux.

Un des problèmes engendrés par la création de caractères définis par l'utilisateur est que le dessin des caractères à la main est fastidieux et que, souvent, le résultat affiché sur l'écran est très différent de l'effet recherché. Le programme de conception de caractères de la cassette de présentation de l'Amstrad comporte un défaut important : il n'affiche pas la définition SYMBOL nécessaire pour recréer un caractère.

Le programme suivant permet de créer des caractères sur l'écran. Il affiche la définition SYMBOL nécessaire à la production d'un caractère et offre la possibilité de sauvegarder ces données dans un fichier. Cela permet de constituer une ''bibliothèque'' de caractères définis par l'utilisateur qui peut servir dans des programmes ultérieurs :

```
10 MODE 1

20 DEFINT c, n, x, y

30 DIM code( 8, 8), symb( 8)

40 INK 3, 20, 6

50 name$="??????"

60 number=240

70 GOSUB 1000

80 GOSUB 3000: GOSUB 4000

90 reponse$=""

100 WHILE reponse$<>"e"

110 newx=x: newy=y
```

```
120 reponse$=LOWER$(INKEY$)

130 IF reponse$="a" AND y>starty THEN newy=y-1

140 IF reponse$="z" AND y<starty+7 THEN newy=y+1

150 IF reponse$="," AND x>startx THEN newx=x-1

160 IF reponse$="." AND x<startx+7 THEN newx=x+1

170 IF newx<>x OR newy<>y THEN GOSUB 2000

180 IF reponse$=" " THEN GOSUB 7000: GOSUB 3000

190 IF reponse$="o" THEN GOSUB 5000: GOSUB 1000: G
OSUB 3000: GOSUB 4000

200 IF reponse$="i" THEN GOSUB 6000: GOSUB 3000: G
OSUB 4000

210 PEN 14

220 LOCATE x,y

230 PRINT CHR$(203);

240 WEND

250 END

1000 CLS

1010 SYMBOL number,0,0,0,0,0,0,0,0

1020 PEN 1

1030 FOR count=1 TO 8

1040 FOR count1=1 TO 8

1050 code(count,count1)=1

1060 NEXT

1070 NEXT

1080 startx=2: starty=2

1090 FOR x=startx TO startx+7

1100 FOR y=starty TO starty+7
```

```
1110 LOCATE x,y

1120 PRINT CHR$(233);

1130 NEXT

1140 NEXT

1150 x=startx: y=starty

1160 RETURN

2000 LOCATE x,y

2010 codex=x-startx+1: codey=y-starty+1

2020 PEN code(codex,codey)

2030 PRINT CHR$(233);

2040 x=newx: y=newy

2050 RETURN

3000 FOR count=1 TO 8

3010 symb$="&X"

3020 FOR count1=1 TO 8

3030 symb$=symb$+MID$(STR$(code(count1,count)-1)
,2,1)

3040 NEXT

3050 symb(count)=VAL(symb$)

3060 NEXT

3070 SYMBOL number,symb(1),symb(2),symb(3),symb(
4),symb(5),symb(6),symb(7),symb(8)

3080 PEN 1

3090 LOCATE 1,starty+10

3100 PRINT"le symbole est: ";CHR$(number);

3110 LOCATE 1,starty+12

3120 PRINT"SYMBOL ";number;symb(1);symb(2);symb(
3);symb(4);symb(5);symb(6);symb(7);symb(8);
```

```
3130 RETURN

4000 PEN 1

4010 LOCATE 1,starty+15

4020 PRINT"nom du symbole: ";name$

4030 LOCATE 1,starty+17

4040 PRINT"numero du symbole: ";number

4050 RETURN

5000 LOCATE 1,21

5010 PEN 1

5020 INPUT"nom du symbole";name$

5030 OPENOUT name$

5040 WRITE #9,name$,number,symb(1),symb(2),symb(
3),symb(4),symb(5),symb(6),symb(7),symb(8)

5050 CLOSEOUT

5060 name$="??????"

5070 number=number+1

5080 RETURN

6000 LOCATE 1,21

6010 PEN 1

6020 INPUT"nom du symbole";name$

6030 OPENIN name$

6040 INPUT #9,name$,number,symb(1),symb(2),symb(
3),symb(4),symb(5),symb(6),symb(7),symb(8)

6050 CLOSEIN

6060 CLS

6070 FOR count=1 TO 8

6080 symb$=BIN$(symb(count))
```

```
6090 length=LEN(symb$)

6100 symb$=STRING$(8-length,"0")+symb$

6110 FOR count1=1 TO 8

6120 LOCATE satrtx+count1-1,starty+count-1

6130 code=VAL(MID$(symb$,count1,1))+1

6140 PEN code

6150 PRINT CHR$(233);

6160 code(count1,count)=code

6170 NEXT

6180 NEXT

6190 RETURN

7000 codex=x-startx+1:codey=y-starty+1

7010 IF code(codex,codey)=1 THEN code(codex,code
y)=2 ELSE code(codex,codey)=1

7020 RETURN
```

EXERCICES

1. Créer un caractère "araignée" et écrire un programme qui redéfinit une touche quelconque du clavier pour générer cette araignée.
2. Écrire un programme pour déplacer l'araignée sur l'écran. On peut utiliser certains sous-programmes de dessin de modèles du chapitre précédent.
3. Améliorer le programme précédent pour générer des araignées dirigées vers le haut, vers le bas, vers la droite et vers la gauche et afficher le caractère correct quand le déplacement est effectué dans une direction particulière.

CARACTÈRES MULTIPLES

En mode 1, un caractère unique n'est pas très grand et il est parfois préférable de créer une figure plus importante à partir de plu-

sieurs formes. Les caractères peuvent être regroupés pour former une chaîne unique si le dessin est complètement horizontal :

```
10 MODE 1
19 REM on definie 3 caracteres
20 SYMBOL 240,0,0,96,96,96,127,18,12
30 SYMBOL 241,0,0,0,0,0,255,0,0
40 SYMBOL 242,248,132,132,255,255,255,72,48
50 lorry$=CHR$(240)+CHR$(241)+CHR$(242)
60 LOCATE 18,13
70 PRINT lorry$
```

Le caractère "camion" peut facilement être placé sous le contrôle du clavier ; pour préserver le réalisme, nous le déplacerons dans une seule direction, vers la droite :

```
10 MODE 1
19 REM on definie 3 caracteres
20 SYMBOL 240,0,0,96,96,96,127,18,12
30 SYMBOL 241,0,0,0,0,0,255,0,0
40 SYMBOL 242,248,132,132,255,255,255,72,48
50 lorry$=CHR$(240)+CHR$(241)+CHR$(242)
70 x=1:y=13
80 LOCATE x,y
90 PRINT lorry$
100 reponse$=""
110 WHILE reponse$<>"e"
120 newx=x
130 reponse$=LOWER$(INKEY$)
140 IF reponse$="." THEN newx=x+1
150 IF newx<>x THEN LOCATE x,y:PRINT" ";lorry$
```

```
160 x=newx

170 WEND
```

Quand le dessin arrive trop près du bord droit, il est automatiquement affiché au début de la ligne suivante. L'Amstrad n'affiche aucun caractère qui déplace le curseur texte en dehors de la fenêtre texte. Si un caractère se trouve dans l'une de ces positions, l'ordinateur déplace le curseur texte sur une position autorisée en respectant les règles suivantes :

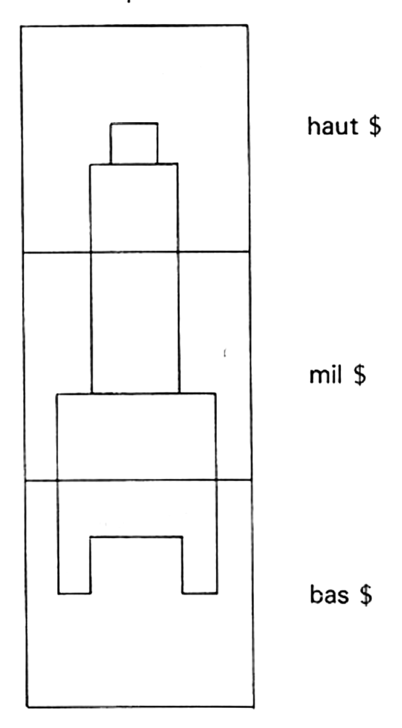
1. Si le curseur dépasse le bord droit de l'écran, il est déplacé sur la première position de la ligne suivante.
2. Si le curseur dépasse le bord gauche de l'écran, il est déplacé sur la dernière position de la ligne précédente.
3. Si le curseur dépasse le bord supérieur de l'écran, l'affichage descend d'une ligne et le curseur reste sur la nouvelle ligne supérieure.
4. Si le curseur dépasse le bord inférieur de l'écran, l'affichage monte d'une ligne et le curseur reste sur la nouvelle ligne inférieure.

Même si seul l'avant du camion se trouve dans une position illégale, l'ordinateur le considère comme une chaîne unique qui se trouve tout entière dans une position non autorisée. En conséquence, dès que le camion atteint une coordonnée texte x 39, qui placerait le premier caractère de la chaîne dans une position de curseur illégale, l'Amstrad affiche *l'ensemble* de la chaîne au début de la ligne suivante. Il ne faut pas oublier cette caractéristique, car cela signifie que la taille d'un caractère multiple impose des restrictions en ce qui concerne les positions sur lesquelles il peut être affiché. Dans cet exemple, la coordonnée texte x maximale est 38.

Code ASCII	Effet
8	Le curseur recule d'un caractère
9	Le curseur avance d'un caractère
10	Le curseur descend d'une ligne
11	Le curseur monte d'une ligne

Figure 2.7 : Les quatre codes ASCII qui génèrent le déplacement du curseur.

L'affichage d'un dessin vertical semble plus difficile, car chaque partie nécessite une instruction LOCATE différente. Dans ce cas, nous profitons de la possibilité d'utiliser les quatre codes ASCII qui ont pour fonction de déplacer le curseur dans certaines directions. En incluant des commandes de déplacement du curseur, on peut décrire une figure verticale dans une seule chaîne. Ces caractères de déplacement du curseur ne sont pas affichés par l'Amstrad mais servent seulement d'instructions de déplacement du curseur texte.



La fusée pourrait être affichée comme une chaîne unique composée de
haut $ + CHR$(8) + CHR$(8) + CHR$(10)

déplace le curseur vers l'arrière
et vers le bas prêt à l'affichage
du caractère suivant

\+ mil $ + CHR$(8) + CHR$(8) + CHR$(10)

même déplacement du curseur

\+ bas $

Figure 2.8 : Caractère vertical multiple créé à l'aide des touches de déplacement du curseur.

La "fusée" peut être affichée sur l'écran à l'aide d'une seule instruction LOCATE :

```
10 MODE 1

20 SYMBOL 240, 0, 24, 24, 24, 24, 36, 36, 36

30 SYMBOL 241, 36, 36, 36, 36, 36, 36, 36, 36

40 SYMBOL 242, 66, 129, 129, 129, 129, 153, 195

50 rocket$=CHR$( 240) +CHR$( 8) +CHR$( 10) +CHR$( 241) +
CHR$( 8) +CHR$( 10) +CHR$( 242)

60 x=20: y=20

70 LOCATE x, y

80 PRINT rocket$

90 reponse$=""

110 WHILE reponse$<>"e" AND y>1

120 newy=y

130 reponse$=LOWER$( INKEY$)

140 IF reponse$="a" THEN newy=y-1

160 IF newy<>y THEN LOCATE x, y+2: PRINT" ": LOCATE
 x, newy: PRINT rocket$

170 y=newy

180 WEND
```

Malheureusement, l'Amstrad a "des habitudes particulières" lorsqu'il traite des chaînes contenant des déplacements de curseur. La figure ci-dessus est parfaitement verticale et aucune de ses extrémités ne se trouve dans une position illégale. Cependant, l'ordinateur la considère comme une chaîne de sept caractères et par conséquent ne permet pas d'afficher la fusée sur une coordonnée texte x supérieure à 34. On peut modifier la ligne 60 de la manière suivante :

```
10 MODE 1

20 SYMBOL 240, 0, 24, 24, 24, 24, 36, 36, 36
```

```
30 SYMBOL 241, 36, 36, 36, 36, 36, 36, 36, 36

40 SYMBOL 242, 66, 129, 129, 129, 129, 153, 195

50 rocket$=CHR$( 240) +CHR$( 8) +CHR$( 10) +CHR$( 241) +
CHR$( 8) +CHR$( 10) +CHR$( 242)

60 x=35: y=20

70 LOCATE x, y

80 PRINT rocket$

90 reponse$=""

110 WHILE reponse$<>"e" AND y>1

120 newy=y

130 reponse$=LOWER$( INKEY$)

140 IF reponse$="a" THEN newy=y-1

160 IF newy<>y THEN LOCATE x, y+2: PRINT" ": LOCATE
 x, newy: PRINT rocket$

170 y=newy

180 WEND
```

On peut utiliser les déplacements du curseur pour créer des figures plus complexes, mais il ne faut pas oublier les restrictions concernant la position d'affichage. Pour toutes les figures, mises à part les plus simples, il est préférable d'employer une série d'instructions LOCATE. Ce petit programme montre les deux possibilités :

```
10 MODE 1

20 SYMBOL 240, 0, 24, 24, 24, 24, 36, 36, 36

30 SYMBOL 241, 36, 36, 36, 36, 36, 36, 36, 36

40 SYMBOL 242, 66, 129, 129, 129, 129, 153, 195

50 rocket$=CHR$( 240) +CHR$( 8) +CHR$( 10) +CHR$( 241) +
CHR$( 8) +CHR$( 10) +CHR$( 242)

60 x=1: y=20
```

```
70 LOCATE 3,23
80 PRINT"caractere definie a l'aide du curseur"
90 FOR ycoord=y TO 1 STEP-1
100 LOCATE x,ycoord+3
110 PRINT" "
120 LOCATE x,ycoord
130 PRINT rocket$
140 NEXT
150 reponse$=""
160 WHILE reponse$=""
170 reponse$=LOWER$(INKEY$)
180 WEND
190 CLS
200 x=36: y=20
210 LOCATE 7,23
220 PRINT"caractere definie avec LOCATE"
230 FOR ycoord=y TO 1 STEP-1
240 LOCATE x,ycoord
250 PRINT CHR$(240)
260 LOCATE x,ycoord+1
270 PRINT CHR$(241)
280 LOCATE x,ycoord+2
290 PRINT CHR$(242)
300 LOCATE x,ycoord+3
310 PRINT" "
320 NEXT
```

On peut obtenir des effets intéressants en définissant des figures légèrement différentes affichées alternativement. Cela peut, par exemple, illustrer la reptation d'un serpent.

```
10 MODE 1
20 SYMBOL 240,0,0,32,80,81,74,130
30 SYMBOL 241,0,0,0,132,74,81,80,32
40 snake1$=CHR$(240)
50 snake2$=CHR$(241)
60 snake$=snake1$
70 y=13
80 FOR x=1 TO 39
90 IF snake$=snake1$ THEN snake$=snake2$ ELSE sn
ake$=snake1$
100 LOCATE x,y
110 PRINT" ";snake$
120 oldtime=TIME
130 WHILE TIME<oldtime+10
140 WEND
150 NEXT
```

De même, on peut définir deux images de camion qui varient légèrement pour donner l'impression que le véhicule est en mouvement, ou bien faire bouger la queue du chien :

```
10 MODE 1
20 SYMBOL 240,0,132,135,132,124,130,130,0
30 SYMBOL 241,0,36,71,132,124,130,65,0
40 dog1$=CHR$(240)
50 dog2$=CHR$(241)
```

```
60 dog$=dog1$
70 y=13
80 FOR x=1 TO 39
90 IF dog$=dog1$ THEN dog$=dog2$ ELSE dog$=dog1$
100 LOCATE x,y
110 PRINT" ";dog$
120 oldtime=TIME
130 WHILE TIME<oldtime+30
140 WEND
150 NEXT
```

EXERCICES

1. Créer une image de bus avec des caractères multiples et la faire se déplacer sur l'écran.
2. Dessiner deux caractères circulaires avec une barre en travers placée différemment sur les deux figures et les afficher alternativement sur l'écran pour donner l'impression d'une roue en mouvement.
3. Ajouter ces roues au bus.

AMÉLIORATION DE LA RÉSOLUTION

On peut concevoir de nombreux jeux amusants en utilisant seulement l'écran texte, mais on est soumis à quelques limites. La meilleure résolution texte est disponible en mode 2, mais celui-ci ne comporte que 25 lignes de 80 caractères. A l'inverse, la moins bonne résolution graphique offre 160 * 200 points. Heureusement, l'Amstrad permet l'affichage de caractères texte sur une position graphique, ce qui permet de générer une meilleure animation et donne à l'utilisateur un contrôle accru dans les jeux. Par ailleurs, cela signifie que les graphiques et les diagrammes peuvent être "labellés" à n'importe quel endroit de l'écran.

Le passage de coordonnées en mode texte aux coordonnées en mode graphique apporte également d'autres changements. Après une commande TAG, les caractères ne peuvent plus être affichés avec une instruction LOCATE. On utilise à la place la commande graphique MOVE pour placer le curseur graphique à l'endroit désiré. L'emplacement du curseur correspond à l'angle supérieur gauche de la grille du caractère texte. Le programme suivant déplace un caractère unique, "envahisseur de l'espace", pour illustrer ce principe.

```
10 MODE 1
20 SYMBOL 240,24,60,126,219,255,255,165,165
30 invader$=CHR$(240)
40 graphicsx=100: graphicsy=200
50 TAG
60 FOR x=graphicsx TO 600
70 MOVE x,graphicsy
80 PRINT" "invader$;
90 NEXT
```

Le point-virgule placé après l'instruction PRINT est essentiel. Une conséquence importante de l'utilisation de l'instruction TAG est que les caractères de contrôle (c'est-à-dire ceux dont les codes ASCII sont compris entre 0 et 31) sont affichés et non exécutés. Voici un programme qui déplace une fusée :

```
10 MODE 1
20 SYMBOL 240,0,24,24,24,24,36,36,36
30 SYMBOL 241,36,36,36,36,36,36,36,36
40 SYMBOL 242,66,129,129,129,129,153,195,129
50 rocket$=CHR$(240)+CHR$(8)+CHR$(10)+CHR$(241)+
CHR$(8)+CHR$(10)+CHR$(242)
60 TAG
70 graphicsx=300: graphicsy=100
```

```
80 FOR y=graphicsy TO 350
90 MOVE graphicsx,y-24
100 PRINT" ";
110 MOVE graphicsx,y
120 PRINT rocket$;
130 NEXT
```

Les problèmes créés par le déplacement du curseur, aussi bien pour le texte que pour le graphisme, nous montrent qu'il est préférable de se limiter à des figures simples. Les figures élaborées à partir d'une série de caractères horizontaux peuvent être affichées après une instruction MOVE unique, parce que le curseur graphique se déplace automatiquement de la largeur d'un caractère après chaque affichage et se trouve donc placé correctement pour générer le caractère suivant. Cela est visible avec le camion composé d'une chaîne unique :

```
10 MODE 1
20 SYMBOL 240,0,0,96,96,96,127,18,12
21 SYMBOL 241,0,0,0,0,0,255,0,0
22 SYMBOL 242,248,132,132,255,255,255,72,48
30 lorry$=CHR$(240)+CHR$(241)+CHR$(242)
40 graphicsx=0: graphicsy=200
50 TAG
60 FOR x=graphicsx TO 600
70 MOVE x,graphicsy
80 PRINT" "lorry$;
90 NEXT
```

La fusée sera affichée grâce à une succession d'instructions MOVE, car chaque caractère est au-dessus du précédent :

```
10 MODE 1
20 SYMBOL 240,0,24,24,24,24,36,36,36
30 SYMBOL 241,36,36,36,36,36,36,36,36
40 SYMBOL 242,66,129,129,129,129,153,195,129
50 rockettop$=CHR$(240)
51 rocketmid$=CHR$(241)
52 rocketbot$=CHR$(242)
60 TAG
70 graphicsx=300: graphicsy=100
80 FOR y=graphicsy TO 350
90 MOVE graphicsx,y-48
100 PRINT" ";
110 MOVE graphicsx,y
120 PRINT rockettop$;
121 MOVE graphicsx,y-16
122 PRINT rocketmid$;
123 MOVE graphicsx,y-32
124 PRINT rocketbot$;
130 NEXT
```

L'instruction TAG peut être annulée dans un programme à l'aide de l'instruction TAGOFF. A la fin d'un programme, elle est automatiquement désactivée.

ACCÉLÉRATION DU DÉROULEMENT

Nous avons remarqué que tout déplacement de figure entraîne un déroulement de programme plus lent. Il existe un certain nombre de manières d'accélérer le déroulement d'un programme.

Tout d'abord, nous pouvons utiliser des nombres entiers chaque fois que c'est possible. Cela permet à l'ordinateur d'exécuter plus rapidement les calculs. L'Amstrad traite tous les nombres comme des réels s'il n'a pas reçu d'instruction contraire. Il est possible de déclarer entières certaines variables grâce à l'instruction DEFINT :

```
1 DEFINT g,x,y
```

L'Amstrad traite maintenant toutes les variables numériques commençant par g, x ou y comme des entiers. La différence de vitesse apparaît lorsque les deux programmes sont comparés :

```
1 DEFINT g, x, y
10 MODE 1
20 SYMBOL 240, 0, 0, 96, 96, 96, 127, 18, 12
21 SYMBOL 241, 0, 0, 0, 0, 0, 255, 0, 0
22 SYMBOL 242, 248, 132, 132, 255, 255, 255, 72, 48
30 lorry$=CHR$( 240) +CHR$( 241) +CHR$( 242)
40 graphicsx=0: graphicsy=200
50 TAG
55 starttime=TIME
60 FOR x=graphicsx TO 600
70 MOVE x, graphicsy
80 PRINT" "lorry$;
90 NEXT
99 totaltime=TIME
100 TAGOFF
110 LOCATE 1, 20
120 PRINT"temps "( totaltime-starttime) /300"secon
des"
```

Une autre méthode pour accélérer le mouvement consiste à prendre note du déplacement minimal que l'Amstrad peut afficher dans chaque mode. Le déplacement horizontal du caractère d'une seule unité en coordonnées x et en mode 0 est si faible que l'affichage a lieu sur le même point. Il faut se déplacer d'au moins quatre unités en mode 0 et de deux unités en mode 1 pour que le déplacement soit visible. La résolution verticale est la même dans les trois modes et le plus petit mouvement visible correspond à une variation de deux unités.

Enfin, il faut entourer les caractères d'une bordure vide pour que le déplacement dans quelque direction que ce soit ne laisse pas de traînées. Le second envahisseur de l'espace de la Figure 2.9 laisse des lignes qui doivent être effacées après chaque déplacement ; cela ralentit le programme.

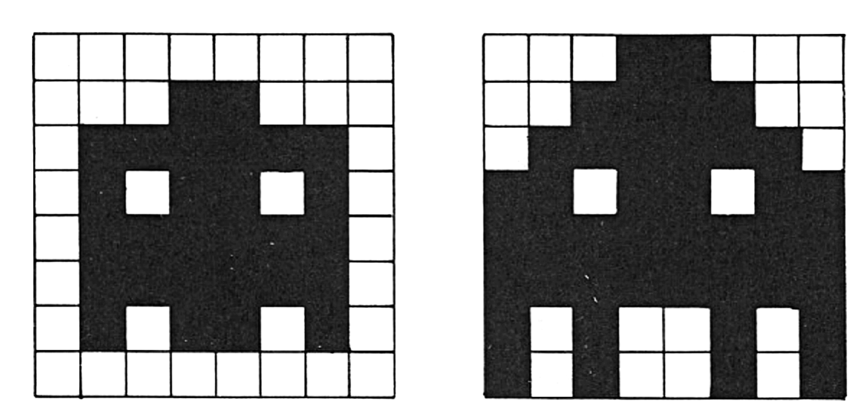


Figure 2.9 : Deux caractères "envahisseur de l'espace" ; le premier est plus utile, car sa bordure efface automatiquement l'image précédente.

POUR TERMINER...

De nombreux jeux identifient les figures par leur couleur. La commande TEST (x,y) permet de déterminer la couleur du stylo qui a été utilisé pour n'importe quelle position graphique de l'écran. En choisissant avec soin la couleur des caractères utilisés dans un jeu, on peut s'assurer que le programme se comportera différemment si on se déplace sur une position contenant un point d'une couleur parti-

culière. Par exemple, toutes les araignées peuvent être roses et, si on essaie de déplacer une mouche sur un point rose, le jeu s'arrête :

```
5 DEFINT c, x, y
10 MODE 0
20 SYMBOL 240, 0, 36, 90, 90, 90, 36, 0, 0
30 SYMBOL 241, 145, 82, 52, 31, 248, 44, 74, 137
40 fly$=CHR$(240)
50 spider$=CHR$(241)
60 GOSUB 1000
70 xfly=300: yfly=200
80 xnew=xfly: ynew=yfly
90 xtest=xfly: ytest=yfly
100 MOVE xfly, yfly: PLOT xfly, yfly, 1
110 GOSUB 2000
120 reponse$="": flydead=0.
130 WHILE reponse$="" OR flydead=0
140 xnew=xfly: ynew=yfly
150 reponse$=LOWER$(INKEY$)
160 IF reponse$="a" THEN ynew=yfly+2: xtest=xnew+
16: ytest=ynew+8
170 IF reponse$="z" THEN ynew=yfly-2: xtest=xnew+
16: ytest=ynew-24
180 IF reponse$="." THEN xnew=xfly+4: xtest=xnew+
48: ytest=ynew-8
190 IF reponse$="," THEN xnew=xfly-4: xtest=xnew-
16: ytest=ynew-8
200 IF xnew<>xfly OR ynew<>yfly THEN GOSUB 2000
210 WEND
```

```
220 MODE 1

230 END

1000 MOVE 0,0

1010 DRAW 0,0,11

1020 TAG

1030 FOR spiders=1 TO 10

1040 spiderx=INT(600*RND(1)+20)

1050 spidery=INT(300*RND(1)+20)

1060 MOVE spiderx,spidery

1070 PRINT spider$;

1080 NEXT

1090 RETURN

2000 colour=TEST(xtest,ytest)

2010 IF colour=11 THEN flydead=1:SOUND 7,2000

2020 xfly=xnew:yfly=ynew

2050 MOVE xfly,yfly

2060 PRINT fly$;

2070 RETURN
```

Ce programme peut être amélioré par l'introduction d'un temps de jeu limité :

```
120 reponse$="":flydead=0

128 oldtime=TIME

130 WHILE (reponse$="" OR flydead=0) AND (xfly<6
00 OR yfly<300)

140 xnew=xfly:ynew=yfly

150 reponse$=LOWER$(INKEY$)
```

```
160 IF reponse$="a" THEN ynew=yfly+2: xtest=xnew+
16: ytest=ynew+8

170 IF reponse$="z" THEN ynew=yfly-2: xtest=xnew+
16: ytest=ynew-24

180 IF reponse$="." THEN xnew=xfly+4: xtest=xnew+
48: ytest=ynew-8

190 IF reponse$="," THEN xnew=xfly-4: xtest=xnew-
16: ytest=ynew-8

200 IF xnew<>xfly OR ynew<>yfly THEN GOSUB 2000

210 WEND

215 TAGOFF: CLS

220 IF flydead=0 THEN PRINT"temps: "TIME-oldtime

230 END
```

Une autre version de la commande TEST est la commande TESTR qui évalue la couleur PEN d'une position relative à la position en cours. Par exemple, TESTR (10, – 5) examine le point qui se trouve dix unités à droite et cinq unités au-dessous du point en cours. Le programme suivant illustre cette instruction :

```
10 MODE 0

20 GOSUB 1000

30 GOSUB 2000

40 END

1000 PAPER 12

1010 PEN 0

1020 CLS

1030 sidex=3: sidey=7

1040 FOR y=sidey TO sidey+11

1050 LOCATE sidex, y

1060 PRINT CHR$(143)CHR$(143);
```

```
1070 LOCATE sidex+15,y
1080 PRINT CHR$(143)CHR$(143);
1090 NEXT
1100 startx=7:starty=5
1109 FOR count=-1 TO 1
1110 LOCATE startx,starty+count
1120 PRINT STRING$(8,CHR$(143));
1121 LOCATE startx,starty-1
1130 LOCATE startx,starty+15+count
1140 PRINT STRING$(8,CHR$(143));
1141 NEXT
1150 TAG
1160 colour=0
1170 leftx=32:rightx=576
1180 boty=150:topy=330
1190 changey=6:changex=32
1200 PLOT leftx+changex,topy+changey,colour
1210 FOR count=1 TO 4
1220 ychange=changey*count
1230 xchange=changex*count
1240 MOVE leftx+xchange,topy+ychange
1250 PRINT CHR$(143);
1260 GOSUB 1600
1280 MOVE rightx-xchange,topy+ychange
1290 PRINT CHR$(143);
1300 GOSUB 1600
```

```
1320 MOVE leftx+xchange, boty-ychange+8
1330 PRINT CHR$(143);
1340 GOSUB 1600
1360 MOVE rightx-xchange, boty-ychange+8
1370 PRINT CHR$(143);
1380 GOSUB 1600
1400 NEXT
1410 SYMBOL 240, 0, 102, 36, 126, 126, 36, 102, 0
1420 SYMBOL 241, 0, 90, 126, 24, 24, 126, 90, 0
1430 side$=CHR$(240)
1440 up$=CHR$(241)
1450 car$=side$
1460 carx=400: cary=338
1470 PLOT carx+16, cary-4, 3
1480 MOVE carx, cary
1490 PRINT car$;
1500 RETURN
1600 FOR count1=1 TO 4
1610 MOVER -32, -16
1620 PRINT CHR$(143);
1630 NEXT
1640 RETURN
2000 carhit=0
2010 changex=0: changey=0
2020 WHILE carhit=0
2030 reponse$=LOWER$(INKEY$)
```

```
2040 IF reponse$="a" THEN changex=0: changey=2: ca
r$=up$: testx=-16: testy=2

2050 IF reponse$="z" THEN changex=0: changey=-2: c
ar$=up$: testx=-16: testy=-16

2060 IF reponse$="." THEN changex=4: changey=0: ca
r$=side$: testx=0: testy=-8

2070 IF reponse$="," THEN changex=-4: changey=0: c
ar$=side$: testx=-36: testy=-8

2080 IF reponse$=" " THEN changex=0: changey=0

2090 IF changex<>0 OR changey<>0 THEN GOSUB 3000

2100 WEND

2110 RETURN

3000 colourpen=TESTR(testx, testy)

3010 IF colourpen<>0 THEN carhit=1: SOUND 7,500

3020 carx=carx+changex: cary=cary+changey

3030 MOVE carx,cary

3040 PRINT car$;

3050 RETURN
```

EXERCICES

1. Ajouter quelques obstacles sur la piste de course : des ballots de paille jaune ou les restes carbonisés d'une voiture.
2. Créer un caractère multiple représentant une chenille verte qui se promène sur l'écran.
3. Ajouter des fraises rouges affichées aléatoirement. Si elle mange une fraise, la chenille devient bleue puis meurt.
4. Ajouter des contrôles à partir du clavier pour guider la chenille en la déplaçant verticalement pour qu'elle évite les fraises. La chenille est sauvée si elle atteint le parterre d'herbe verte placé à droite de l'écran.

3

COURBES ET DIAGRAMMES

Dans les chapitres précédents, nous avons abordé l'aspect ludique du graphisme, mais il existe des applications plus sérieuses, même sur un micro-ordinateur. Ces dernières années, nous avons assisté à une prolifération de logiciels sophistiqués adaptés aux problèmes de gestion simples. Le but de la plupart de ces programmes est la présentation de l'information d'une façon plus compréhensible. Au lieu de donner des listes sans fin de chiffres et de résultats, le logiciel génère à partir des données manipulées des courbes, des histogrammes et des diagrammes sectoriels ou bien la combinaison des trois. L'emploi de la couleur et du graphisme haute résolution facilite l'affichage des résultats et met en valeur certaines particularités. L'avantage supplémentaire présenté par l'ordinateur est qu'il peut rapidement refaire les calculs et afficher une nouvelle courbe ou un nouvel histogramme pour illustrer par exemple une chute des chiffres des ventes.

Nous allons étudier dans ce chapitre un programme qui génère les trois formes les plus familières de la représentation graphique de données : les courbes, les histogrammes et les diagrammes sectoriels. A ce stade, il est important de connaître certaines des règles qui doivent être utilisées pour la conception d'un programme.

Tout d'abord, il est déconseillé d'écrire un programme constitué d'un seul bloc. Il est plus facile de développer, de "débugger" et d'améliorer un programme s'il est constitué de modules, c'est-à-dire de petites sections ayant chacune une tâche spécifique à exécuter, comme par exemple le tracé des axes d'une courbe ou la coloration des classes d'un histogramme.

Ensuite, un programme est très rigide s'il est associé à des valeurs spécifiques. Un programme qui trace les axes d'une courbe particulière fonctionne très bien mais contient des lignes telles que :

```
100 MOVE 308,390
110 DRAW 308,120
120 DRAW 630,120
```

Ces lignes sont difficiles à réutiliser et l'élaboration d'une nouvelle courbe peut nécessiter la réécriture du programme. Il est préférable d'utiliser si possible des *variables*. L'emploi de variables présente de nombreux avantages : les noms de variables sont plus significatifs que les séries de nombres dans les commandes MOVE et DRAW et le programme est plus facile à comprendre et à modifier, surtout s'il a été abandonné pendant quelque temps.

De plus, l'utilisation de variables signifie que l'on peut écrire un programme principal qui génère une courbe pour n'importe quel jeu de données ; les seules modifications nécessaires sont relatives aux valeurs des données et il n'est pas indispensable de modifier une par une les lignes de programme.

L'écriture d'un programme de ce type demande plus de réflexion au départ et peut-être un développement un peu plus long, mais cela en vaut la peine. Cependant, cette approche évite l'écriture de plusieurs programmes qui exécutent des tâches identiques.

GRAPHISME AVEC DES LIGNES ET DES POINTS

La première question qui se pose lors du tracé de courbes est le choix du type de résolution. Sur l'Amstrad, nous pouvons choisir un mode quelconque parmi trois modes qui ont la même résolution verticale et des résolutions horizontales différentes. En général, il est préférable de tracer une courbe comportant de nombreux points en mode 2 pour tirer un meilleur parti de la haute résolution. Malheureusement, en mode 2 nous sommes limités à deux couleurs. Si le nombre de points à tracer est inférieur à 300, le mode 1 est un compromis raisonnable entre les exigences liées à la résolution et à la couleur : il offre 320 points adressables horizontalement avec un choix de quatre couleurs.

L'Amstrad a été conçu de telle sorte qu'un changement de mode n'affecte pas les coordonnées graphiques. Par conséquent, le programme qui suit se déroule d'une manière identique dans n'importe lequel des trois modes. Par contre, la résolution varie.

Nous allons commencer par développer un petit programme. Pour le moment, nous utiliserons la totalité de l'écran pour le tracé de la courbe et nous laisserons de côté l'écriture des labels. Ce programme est constitué d'une série de sous-programmes ; cela nous donne plus de souplesse et permet de tracer plusieurs jeux de données sur une seule courbe ou de dessiner des courbes multiples sans altération majeure du programme :

```
10 MODE 1

20 GOSUB 5CO

30 GOSUB 800
```

```
40 END

499 REM trace des axes

500 ypoints=399

510 xpoints=639

520 MOVE 0,ypoints

530 DRAW 0,0,1

540 DRAW xpoints,0

550 minx=200

560 maxx=4000

570 diffx=maxx-minx

580 miny=100

590 maxy=1000

600 diffy=maxy-miny

610 pointx=diffx/xpoints

620 pointy=diffy/ypoints

690 RETURN

800 READ noofpoints

805 DIM x(noofpoints),y(noofpoints)

810 FOR count=1 TO noofpoints

815 READ x(count):xdispl=(x(count)-minx)
/pointx

820 READ y(count):ydispl=(y(count)-miny)
/pointy

825 PLOT xdispl,ydispl

830 NEXT

990 RETURN

1000 DATA 5,200,100,1000,200,1500,300,25
00,600,4000,1000
```

La ligne 520 trace les axes de la courbe. L'origine est placée dans l'angle inférieur gauche de l'écran. Il est nécessaire de connaître les valeurs maximale et minimale des données pour que l'échelle de la courbe soit correcte (c'est-à-dire que tous les points se trouvent sur l'écran). Ces valeurs sont établies explicitement lignes 550 à 600, bien que l'ordinateur puisse les calculer à partir des données fournies. Une fois que l'Amstrad a trouvé la différence entre les valeurs maximale et minimale, il peut calculer (lignes 610 et 620) la valeur des unités sur les axes X et Y pour que l'ensemble des données apparaisse sur l'écran. L'origine représente le point de coordonnées minimal (le point de coordonnées maximal est placé dans l'angle supérieur droit).

Enfin, le programme lit les coordonnées des données, les modifie en fonction de l'échelle, puis trace les points (lignes 800 à 830). Dans ce cas, les données sont déjà classées sur l'axe des X, des plus petites aux plus grandes. L'ordre aléatoire des données sera traité ultérieurement.

On peut faire tourner plusieurs fois le programme avec des données différentes pour max x et max y pour visualiser l'effet obtenu. Si on double max x, la courbe est "contractée" vers la gauche car le programme réserve de la place pour les valeurs de x plus importantes qui peuvent se trouver dans les données. Si on double max y, la courbe est contractée vers le bas. Les variables min x et min y ont des effets similaires si elles sont modifiées. Si les coordonnées de l'origine (100,200) ne conviennent pas et que l'on préfère (0,0), il faut attribuer à min x et à min y la valeur 0. On remarque qu'une partie de l'écran est inutilisée parce qu'il n'y a pas de points dans cette zone. Il ne faut pas oublier que les données de la ligne 1000 font partie d'une certaine gamme de valeurs et que la modification de min x peut entraîner la perte de certains points.

Le déroulement du programme révèle quelques problèmes ; la courbe et les points sont assez difficiles à distinguer. Certaines modifications peuvent améliorer cet état de fait.

```
10 MODE 1

20 GOSUB 500

30 GOSUB 800

40 END
```

```
499 REM trace des axes

500 ypoints=399

510 xpoints=639

520 MOVE 0,ypoints

530 DRAW 0,0,1

540 DRAW xpoints,0

550 minx=200

560 maxx=4000

570 diffx=maxx-minx

580 miny=100

590 maxy=1000

600 diffy=maxy-miny

610 pointx=diffx/xpoints

620 pointy=diffy/ypoints

690 RETURN

800 READ noofpoints

805 DIM x(noofpoints),y(noofpoints)

806 READ pencolour,papercolour

807 INK 0,papercolour

808 INK 1,pencolour

809 PAPER 0:PEN 1

810 FOR count=1 TO noofpoints

815 READ x(count):xdispl=(x(count)-minx)
/pointx

820 READ y(count):ydispl=(y(count)-miny)
/pointy

825 PLOT xdispl,ydispl
```

```
830 NEXT

990 RETURN

1000 DATA 5, 0, 24, 200, 100, 1000, 200, 1500, 3
00, 2500, 600, 4000, 1000
```

Les couleurs utilisées pour tracer la courbe sont maintenant spécifiées dans l'instruction DATA. On pourrait même aller plus loin et spécifier un autre mode ; mais restons en mode 1. Les points peuvent être mis plus en valeur en dessinant sur chaque position une croix ou un carré, ce qui est facilement réalisé grâce au déplacement relatif :

```
10 MODE 1

20 GOSUB 500

30 GOSUB 800

40 END

499 REM trace des axes

500 ypoints=399

510 xpoints=639

520 MOVE 0,ypoints

530 DRAW 0,0,1

540 DRAW xpoints,0

550 minx=200

560 maxx=4000

570 diffx=maxx-minx

580 miny=100

590 maxy=1000

600 diffy=maxy-miny

610 pointx=diffx/xpoints

620 pointy=diffy/ypoints
```

```
690 RETURN

800 READ noofpoints

805 DIM x(noofpoints),y(noofpoints)

806 READ pencolour,papercolour

807 INK 0,papercolour

808 INK 1,pencolour

809 PAPER 0:PEN 1

810 crossx=10

811 crossy=10

812 flag=1

814 FOR count=1 TO noofpoints

815 READ x(count):xdispl=(x(count)-minx)
/pointx

820 READ y(count):ydispl=(y(count)-miny)
/pointy

825 PLOT xdispl,ydispl

830 MOVER -crossx,crossy

840 DRAWR 2*crossx,-2*crossy

850 MOVER -2*crossx,0

860 DRAWR 2*crossx,2*crossy

870 MOVER -crossx,-crossy

880 NEXT

990 RETURN

1000 DATA 5,0,24,200,100,1000,200,1500,3
00,2500,600,4000,1000
```

Le déplacement qui crée une branche de la croix est défini par deux constantes et non pas par les données, parce que la taille de ce genre de marqueur reste en général fixe au cours des différents déroule-

ments du programme. Il est possible d'agrandir ou de diminuer la taille de la croix, il suffit pour cela de modifier deux lignes.

Le programme sera plus utile s'il comporte une option permettant de joindre deux points. Cela peut être indiqué par l'attribution d'une variable "drapeau" à l'une des deux valeurs qui indique si les points doivent être tracés séparément ou conjointement :

```
10 MODE 1
20 GOSUB 500
30 GOSUB 800
40 END
499 REM trace des axes
500 ypoints=399
510 xpoints=639
520 MOVE 0,ypoints
530 DRAW 0,0,1
540 DRAW xpoints,0
550 minx=200
560 maxx=4000
570 diffx=maxx-minx
580 miny=100
590 maxy=1000
600 diffy=maxy-miny
610 pointx=diffx/xpoints
620 pointy=diffy/ypoints
690 RETURN
800 READ noofpoints
805 DIM x(noofpoints),y(noofpoints)
806 READ pencolour,papercolour
```

```
807 INK 0,papercolour

808 INK 1,pencolour

809 PAPER 0:PEN 1

810 crossx=10

811 crossy=10

812 flag=1

814 FOR count=1 TO noofpoints

815 READ x(count):xdispl=(x(count)-minx)
/pointx

816 x(count)=xdispl

820 READ y(count):ydispl=(y(count)-miny)
/pointy

821 y(count)=ydispl

825 PLOT xdispl,ydispl

830 MOVER -crossx,crossy

840 DRAWR 2*crossx,-2*crossy

850 MOVER -2*crossx,0

860 DRAWR 2*crossx,2*crossy

870 MOVER -crossx,-crossy

875 IF flag=1 AND count>1 THEN DRAW x(co
unt-1),y(count-1)

880 NEXT

990 RETURN

1000 DATA 5,0,24,200,100,1000,200,1500,3
00,2500,600,4000,1000
```

L'inconvénient majeur du programme est l'absence de labels. Étant donné que la courbe commence à partir de l'angle inférieur gauche de l'écran, il n'y a pas de place pour le label et il semble que le pro-

gramme nécessite des modifications importantes. En fait, l'utilisation de variables et les capacités de l'Amstrad à modifier l'origine des courbes rendent le déplacement des axes très facile à réaliser :

```
10 MODE 1
20 GOSUB 500
30 GOSUB 800
40 END
499 REM trace des axes
500 ox=100: oy=50
505 ORIGIN ox, oy
510 ypoints=399-oy
515 xpoints=639-ox
520 MOVE 0, ypoints
530 DRAW 0, 0, 1
540 DRAW xpoints, 0
550 minx=200
560 maxx=4000
570 diffx=maxx-minx
580 miny=100
590 maxy=1000
600 diffy=maxy-miny
610 pointx=diffx/xpoints
620 pointy=diffy/ypoints
690 RETURN
800 READ noofpoints
805 DIM x( noofpoints), y( noofpoints)
806 READ pencolour, papercolour
```

```
807 INK 0,papercolour

808 INK 1,pencolour

809 PAPER 0:PEN 1

810 crossx=10

811 crossy=10

812 flag=1

814 FOR count=1 TO noofpoints

815 READ x(count):xdispl=(x(count)-minx)
/pointx

816 x(count)=xdispl

820 READ y(count):ydispl=(y(count)-miny)
/pointy

821 y(count)=ydispl

825 PLOT xdispl,ydispl

830 MOVER -crossx,crossy

840 DRAWR 2*crossx,-2*crossy

850 MOVER -2*crossx,0

860 DRAWR 2*crossx,2*crossy

870 MOVER -crossx,-crossy

875 IF flag=1 AND count>1 THEN DRAW x(co
unt-1),y(count-1)

880 NEXT

990 RETURN

1000 DATA 5,0,24,200,100,1000,200,1500,3
00,2500,600,4000,1000
```

La ligne 500 doit être modifiée, puisque l'on n'utilise plus la totalité de l'écran pour la courbe. Le reste du programme n'a pas à être modifié : toutes les lignes dessinées et tous les points tracés sont relatifs à la nouvelle origine, comme on le voit lors du déroulement du pro-

gramme. Si on attribue à ox et oy de nouvelles valeurs, on s'aperçoit que la forme de la courbe reste la même, quelle que soit l'origine et indépendamment de sa taille.

Nous avons maintenant assez de place pour marquer les intervalles sur les axes et leur attribuer des labels. Cela ne peut pas être fait de façon complètement automatique. L'attribution de labels sur l'axe des Y ne doit pas poser de problème, car nous avons choisi une origine qui laisse un espace suffisant.

Le nombre maximal d'intervalles est limité par deux facteurs : la résolution du mode et la largeur des caractères à afficher au-dessous de chaque graduation. L'ordinateur peut déterminer si l'intervalle choisi est trop faible et entraîne le chevauchement des caractères lors de l'écriture des labels ; mais il ne peut pas réellement sélectionner l'intervalle qui convient. L'utilisateur choisit généralement les intervalles suivants : .5, 10, 20, 25, 100 etc. L'Amstrad rejette les valeurs impossibles.

L'attribution de labels se fait en deux temps : graduation des axes et affichage des caractères. La première étape n'est pas exécutée si les caractères à afficher se chevauchent. Il faut indiquer à l'ordinateur la taille maximale des chaînes à afficher pour qu'il puisse calculer si elles sont affichables :

```
504 INK 0,24:INK 1,24

630 xrange=5

640 yrange=10

650 xwidth=INT(xpoints/xrange)

660 xheight=INT(ypoints/yrange)

670 maxxstring=4

675 charwidth=16

676 charheight=16

680 graphxstring=charwidth*maxxstring

690 IF xwidth<graphxstring OR xwidth<cha
rwidth THEN RETURN

692 maxystring=4
```

```
694 graphystring=charwidth*maxystring

696 IF ox<graphystring OR yheight<charhe
ight THEN RETURN

700 xtick=6

702 ytick=8

704 xvalue=diffx/xrange

706 TAG

708 FOR count=0 TO xrange

710 MOVE 0,0

712 MOVER xwidth*count,0

714 DRAWR 0,-xtick

728 NEXT

730 yvalue=diffy/yrange

732 FOR count=0 TO yrange

734 MOVE 0,0

736 MOVER 0,yheight*count

738 DRAWR -ytick,0

754 NEXT

790 RETURN
```

Comme pour les points mis en valeur par des croix, il est possible d'interférer sur la qualité du graphisme des graduations. On peut modifier leur taille en corrigeant les lignes appropriées.

Au lieu de recalculer les positions des graduations lors de l'affichage des valeurs x et y, on peut insérer le sous-programme d'affichage au moment du dessin des graduations :

```
708 FOR count=0 TO xrange

710 MOVE 0,0

712 MOVER xwidth*count,0
```

```
714 DRAWR 0,-xtick

716 MOVER -charwidth/2,-xtick

718 number$=STR$(minx+count*xvalue)

720 number$=MID$(number$,2,maxxstring)

722 length=LEN(number$)

724 IF MID$(number$,length)="." THEN num
ber$=MID$(number$,1,length-1)

726 PRINT number$;

728 NEXT

730 yvalue=diffy/yrange

732 FOR count=0 TO yrange

734 MOVE 0,0

736 MOVER 0,yheight*count

738 DRAWR -ytick,0

740 MOVER -graphystring,charheight/2

742 number$=STR$(miny+count*yvalue)

744 number$=MID$(number$,2,maxystring)

746 length=LEN(number$)

748 IF MID$(number$,length)="." THEN num
ber$=MID$(number$,1,length-1)

750 IF LEN(number$)<maxystring THEN numb
er$=STRING$(maxystring-LEN(number$)," ")
+number$

752 PRINT number$;

754 NEXT

790 RETURN
```

On attribue également des labels aux axes et on donne un titre à

la courbe. Il faut vérifier à nouveau que l'origine a été choisie de telle sorte qu'il y ait de la place pour l'affichage des caractères :

```
756 xlabel$="population des souris"

758 ylabel$="fromage consome"

760 xlablength=LEN(xlabel$)*charwidth

762 xlabstart=(xpoints-xlablength)/2

764 MOVE xlabstart,-2*charheight

766 PRINT xlabel$;

768 ylablength=LEN(ylabel$)*charheight

770 ylabstart=(ypoints+ylablength)/2

772 FOR count=1 TO LEN(ylabel$)

774 char$=MID$(ylabel$,count,1)

776 MOVE -charwidth*(maxystring+2),ylabs
tart-charheight*(count-1)

778 PRINT char$;

780 NEXT

790 RETURN
```

Le programme est incomplet, puisqu'il ne travaillera que sur des données préalablement ordonnées. Cela n'est valable que pour certaines opérations, comme par exemple le calcul des précipitations sur une certaine période ou des fluctuations d'un compte bancaire sur plusieurs mois (on a alors essentiellement besoin d'un axe négatif). Si les valeurs doivent être triées par ordre croissant, il ne suffit pas de pouvoir les lire, de déterminer leur échelle et de les tracer. Elles doivent d'abord être stockées et comparées, puis éventuellement réorganisées. Toutes les données sont lues dans un tableau puis triées :

```
814 FOR count=1 TO noofpoints

815 READ x(count):xdispl=(x(count)-minx)
/pointx

816 x(count)=xdispl
```

```
820 READ y(count):ydispl=(y(count)-miny)
/pointy

821 y(count)=ydispl

825 NEXT

830 FOR count=2 TO noofpoints

840 FOR value=count TO 2 STEP -1

850 IF x(value)<x(value-1) THEN GOSUB 15
00 ELSE value=2

860 NEXT

870 NEXT

1500 tempx=x(value):tempy=y(value)

1510 x(value)=x(value-1):y(value)=y(valu
e-1)

1520 x(value-1)=tempx:y(value-1)=tempy

1530 RETURN
```

Quelques autres lignes doivent être modifiées, puisque les coordonnées ne sont plus directement lues et tracées à partir des données.

Le tri effectué est appelé *tri d'insertion*. Les deux premières coordonnées x sont ordonnées, puis la troisième est insérée en bonne place. Le processus est ensuite répété pour l'insertion de la quatrième coordonnée et ainsi de suite jusqu'à ce que toutes les coordonnées soient placées par ordre croissant.

Le tri de variables numériques et de chaînes est très fréquent en informatique. Si plusieurs centaines de nombres sont concernées, le processus ci-dessus est un peu long et on peut en utiliser un autre ; par exemple, le tri par bulle est particulièrement adapté si les données sont déjà en partie classées. Si la quantité de données est trop importante, il faut avoir recours à un sous-programme de tri en code machine. Lorsque les données sont triées en fonction de l'abscisse, l'ordinateur peut calculer seul les valeurs de min x, min y, max x et max y :

```
880 FOR count=1 TO noofpoints

890 PLOT x(count),y(count)
```

```
900 MOVER -crossx, crossy

910 DRAWR 2*crossx, -2*crossy

920 MOVER -2*crossx, 0

930 DRAWR 2*crossx, 2*crossy

940 MOVER -crossx, -crossy

960 IF flag=1 AND count>1 THEN DRAW x(co
unt-1), y(count-1)

970 NEXT

990 RETURN

1000 DATA 5, 0, 24, 1000, 200, 4000, 1000, 200,
100, 2500, 600, 1500, 300
```

Le programme a encore quelques faiblesses, que l'on peut essayer de rectifier dans les exercices suivants ; mais il est adapté à la plupart des situations.

EXERCICES

1. Quelles modifications sont nécessaires pour que le programme tourne en mode 0 ou en mode 2 ? Quelles sont les valeurs qui varient en fonction du mode et qui pourraient être remplacées par des variables ?
2. Pour l'instant, l'axe des y est toujours dessiné à gauche de la courbe et l'axe des x à sa base. Modifier le programme pour que les axes x et y passent par le point (0,0) quand les données incluent des coordonnées positives et négatives.
3. Modifier le programme pour que des jeux de données successifs puissent être tracés sur la même courbe dans des couleurs différentes. Ne pas répéter des sections du programme pour dessiner d'autres lignes : identifier les parties du programme nécessaires et les appeler sous la forme de sous-programmes.
4. Modifier le programme pour que deux courbes d'échelles différentes soient affichées dans les moitiés supérieure et inférieure de l'écran (il faut modifier les ordonnées et changer deux fois l'origine).
5. Modifier le programme pour que les données de la courbe puissent être lues à partir d'un fichier ou sauvegardées.

HISTOGRAMME

Le tracé d'un histogramme est plus facile à réaliser à partir du programme de dessin de courbe, car la plupart des problèmes ont été résolus. Le processus est simplifié puisque nous ne traitons que les coordonnées y ; dès que nous connaissons le nombre de données, il est facile de calculer la largeur de chaque classe. Le programme précédent a mis en évidence l'avantage de l'utilisation généralisée de variables et de sous-programmes :

```
10 MODE 1
20 GOSUB 500
30 GOSUB 800
40 GOTO 40
499 REM trace des axes
500 ox=100: oy=50
505 ORIGIN ox, oy
510 ypoints=399-oy
515 xpoints=639-ox
520 MOVE 0, ypoints
530 DRAW 0, 0, 1
540 DRAW xpoints, 0
550 minx=200
560 maxx=4000
570 diffx=maxx-minx
580 miny=100
590 maxy=1000
600 diffy=maxy-miny
620 pointy=diffy/ypoints
```

```
630 READ noofbars: xrange=noofbars

640 yrange=9

650 xwidth=INT( xpoints/xrange)

660 yheight=INT( ypoints/yrange)

675 charwidth=16

676 charheight=16

692 maxystring=4

694 graphystring=charwidth*maxystring

696 IF ox<graphystring OR yheight<charhe
ight THEN RETURN

700 xtick=6

702 ytick=8

706 TAG

710 MOVE 0,0

712 MOVER xwidth*count,0

714 DRAWR 0,-xtick

716 MOVER -charwidth/2,-xtick

718 number$=STR$( minx+count*xvalue)

720 number$=MID$( number$, 2, maxxstring)

722 length=LEN( number$)

730 yvalue=diffy/yrange

732 FOR count=0 TO yrange

734 MOVE 0,0

736 MOVER 0,yheight*count

738 DRAWR -ytick,0

740 MOVER -graphystring,charheight/2
```

```
742 number$=STR$(miny+count*yvalue)

744 number$=MID$(number$,2,maxystring)

746 length=LEN(number$)

748 IF MID$(number$,length)="." THEN num
ber$=MID$(number$,1,length-1)

750 IF LEN(number$)<maxystring THEN numb
er$=STRING$(maxystring-LEN(number$)," ")
+number$

752 PRINT number$;

754 NEXT

756 xlabel$="population des souris"

758 ylabel$="fromage consome"

760 xlablength=LEN(xlabel$)*charwidth

762 xlabstart=(xpoints-xlablength)/2

764 MOVE xlabstart,-2*charheight

766 PRINT xlabel$;

768 ylablength=LEN(ylabel$)*charheight

770 ylabstart=(ypoints+ylablength)/2

772 FOR count=1 TO LEN(ylabel$)

774 char$=MID$(ylabel$,count,1)

776 MOVE -charwidth*(maxystring+2),ylabs
tart-charheight*(count-1)

778 PRINT char$;

780 NEXT

790 RETURN

800 DIM y(noofbars)

806 READ pencolour,papercolour

807 INK 0,papercolour
```

```
808 INK 1,pencolour

809 PAPER 0:PEN 1

814 FOR count=1 TO noofbars

815 READ y

820 y(count)=(y-miny)/pointy

825 NEXT

830 barwidth=xwidth-4

840 FOR count=1 TO noofbars

850 MOVE 0,0

860 DRAWR 0,y(count),1

870 DRAWR barwidth,0

880 DRAWR 0,-y(count)

890 ox=ox+xwidth

900 ORIGIN ox,oy

910 NEXT

990 RETURN

1000 DATA 10,0,24,350,190,760,440,990,12
4,846,545,666,222
```

Le sous-programme de la ligne 800 trace chaque classe comme une série de déplacements relatifs en fonction de l'origine. Si on déplace l'origine d'une distance déterminée entre le dessin de chaque classe, on peut utiliser une boucle pour tracer une succession de classes.

L'affichage final est plus significatif si les classes ont différentes couleurs. Malheureusement, l'Amstrad ne comporte pas de commandes qui permettent de colorer une zone graphique et les classes doivent être remplies par de simples lignes qu'il faut tracer le plus rapidement possible. Dans le dernier chapitre, nous avons identifié plusieurs méthodes pour accélérer le déroulement d'un programme et nous pouvons maintenant utiliser cette possibilité :

```
830 barwidth=xwidth-4
835 colour=2
840 FOR count=1 TO noofbars
845 IF colour=3 THEN colour=2 ELSE colou
r=3
850 FOR bar=0 TO barwidth STEP charwidth
/8
860 MOVE 0+bar,0
870 DRAWR 0,y(count),colour
880 NEXT
890 ox=ox+xwidth
900 ORIGIN ox,oy
910 NEXT
990 RETURN
```

Les lignes sont dessinées verticalement et non horizontalement, parce que la plupart des classes sont plus hautes que larges ; cela signifie qu'il faut moins de lignes verticales pour les remplir. La seule méthode de coloration rapide est réalisée avec des sous-programmes en langage machine.

Il est également possible de tracer les histogrammes en donnant de la perspective aux différentes classes :

```
830 barwidth=xwidth-4
834 barside=barwidth/4:bartop=barside
835 colour=2
840 FOR count=1 TO noofbars
845 IF colour=3 THEN colour=2 ELSE colou
r=3
846 bartopcount=0
```

```
850 FOR bar=0 TO barwidth STEP charwidth
/8

860 PLOT 0+bar,0,1

870 DRAWR 0,y(count)+bartopcount,colour

872 PLOTR 0,0,1

875 bartopcount=bartopcount+charwidth/8:
IF bartopcount>bartop THEN bartopcount=b
artop

880 NEXT

882 y=y(count):MOVE 0,0

883 DRAWR 0,y,1

884 DRAWR barwidth,0

885 DRAWR 0,-y

886 MOVER 0,y

887 DRAWR barside,bartop

888 DRAWR 0,-y-bartop

890 ox=ox+xwidth

900 ORIGIN ox,oy

910 NEXT

990 RETURN
```

EXERCICES

1. Effectuer les modifications nécessaires pour que le programme tourne correctement en mode 0 et en mode 2..
2. Modifier le programme pour qu'il trace un histogramme horizontal et non vertical.
3. Modifier l'histogramme en lui donnant "trois dimensions" pour que chaque jeu de données soit dessiné devant le précédent.

Un diagramme sectoriel a peu de choses en commun avec les courbes et les histogrammes et nous devons développer un programme différent, qui sera composé de certains des sous-programmes précédents. La résolution est importante pour le diagramme sectoriel qui nécessite le tracé précis d'un cercle, mais l'emploi de la couleur donne à ce diagramme plus d'impact. Nous devons donc trouver un compromis et utiliser le mode 1.

Le programme qui génère le premier diagramme sectoriel dessine simplement un cercle et le divise en secteurs de tailles appropriées et de couleurs différentes :

```
10 MODE 1

20 GOSUB 1000

30 GOSUB 2000

40 GOSUB 3000

50 END

999 REM

1000 READ centrex,centrey

1010 READ radius

1020 READ numberofvalues

1030 DIM value(numberofvalues),angle(num
berofvalues)

1040 totalofvalues=0

1050 FOR count=1 TO numberofvalues

1060 READ value(count)

1070 totalofvalues=totalofvalues+value(c
ount)

1080 NEXT

1090 RETURN

1100 DATA 200,200,120,4,1,2,3,4
```

```
1999 REM calcul de l'angle pour chaque s
ecteur

2000 FOR count=1 TO numberofvalues

2010 angle(count)=2*PI*value(count)/tota
lofvalues

2020 NEXT

2030 RETURN

3000 startangle=0

3020 stepsize=PI/60

3030 colour=1

3040 noofcolours=3

3050 FOR count=1 TO numberofvalues

3060 endangle=startangle+angle(count)

3069 REM on change de couleur pour chaqu
e secteur

3070 colour=1+(colour+1)MOD noofcolours

3080 IF count=numberofvalues AND numbero
fvalues MOD noofcolours=1 THEN colour=1+
(colour+1)MOD noofcolours

3090 MOVE centrex,centrey: DRAW centrex+r
adius*SIN(startangle),centrey+radius*COS
(startangle),colour

3099 REM dessin secteur

3100 FOR angle=startangle TO endangle ST
EP stepsize

3110 DRAW centrex+radius*SIN(angle),cent
rey+radius*COS(angle)

3120 NEXT

3130 startangle=endangle

3140 NEXT

3150 RETURN
```

Le sous-programme de la ligne 1000 lit les valeurs à partir de l'instruction DATA et calcule leur somme. Cela est nécessaire pour déterminer l'angle du secteur correspondant dans le sous-programme de la ligne 2000. Chaque secteur est dessiné dans une couleur différente dans le sous-programme de la ligne 3000.

Le déroulement du programme révèle que, pour un diagramme sectoriel avec un rayon important, le tracé est assez lent. On peut accélérer le processus en prenant comme nouvelle origine le centre du cercle ; cela rend l'exécution des calculs plus rapide :

```
3000 ORIGIN centrex, centrey

3099 REM dessin secteur

3100 FOR angle=startangle TO endangle ST
EP stepsize

3110 DRAW radius*SIN(angle), radius*COS(a
ngle)

3120 NEXT

3190 MOVE 0,0: DRAW radius*SIN(startangle
), radius*COS(startangle), colour
```

La nécessité de calculer le sinus et le cosinus des angles ralentit le processus, mais elle est impérative. Cependant, pour démontrer qu'il existe plusieurs façons de procéder, nous allons étudier un programme plus rapide. Il ne calcule qu'un seul sinus et un seul cosinus puis utilise ces valeurs pour déterminer le point de la circonférence suivant :

```
2000 radval=radius*4

2005 FOR count=1 TO numberofvalues

2010 angle(count)=radval*value(count)/to
talofvalues

2020 NEXT

2030 RETURN

3000 ORIGIN centrex, centrey

3009 REM on utlise des entiers dans les
boucles afin d'etre plus rapide
```

```
3010 DEFINT c

3030 colour=1

3035 thesin=SIN(2*PI/radval):thecos=COS(
2*PI/radval)

3037 x1=radius:y1=0

3039 REM mode 0 on change la couleur des
 points

3040 noofcolours=3

3050 FOR count=1 TO numberofvalues

3069 REM changement de couleur par secte
ur

3070 colour=1+(colour+1)MOD noofcolours

3079 REM 1er et dernier secteur de coule
ur differente

3080 IF count=numberofvalues AND numbero
fvalues MOD noofcolours=1 THEN colour=1+
(colour+1)MOD noofcolours

3099 REM dessin des secteurs

3100 FOR count1=1 TO angle(count)

3110 x=x1*thecos-y1*thesin

3112 y=x1*thesin+y1*thecos

3114 PLOT x,y,colour

3116 x1=x:y1=y

3120 NEXT

3129 REM on centre ce secteur

3130 MOVE 0,0:DRAW x,y

3140 NEXT

3150 RETURN
```

Le diagramme sectoriel est plus significatif si chaque secteur est coloré différemment, et nous allons modifier le premier programme en conséquence. La méthode la plus simple consiste à dessiner des lignes successives à partir du centre jusqu'à chaque point de la circonférence :

```
1 REM du programme precedent on change

2 REM les lignes :

3020 stepsize=PI/500

3109 REM on part du centre

3110 MOVE 0,0: DRAW radius*SIN(angle),rad
ius*COS(angle)
```

Ce programme est très lent mais malheureusement, si les pas d'itérations sont importants, certains pixels ne sont pas pris en compte et gardent la couleur du fond. On peut accélérer le déroulement en colorant alternativement des secteurs (les autres conserveront la couleur du fond). Cette approche permet de gagner du temps si les valeurs sont lues dans un tableau puis triées par ordre croissant. En réorganisant l'ordre dans lequel les secteurs sont tracés, il est possible de ne colorer que les petits secteurs. Cependant, cela est contraire au choix du mode 1 ; par ailleurs, l'attribution de labels aux secteurs est primordiale pour la compréhension du diagramme sectoriel.

On peut accélérer le déroulement du programme en calculant les coordonnées de la circonférence et en les stockant dans un tableau puis en utilisant les valeurs du tableau lors du dessin du diagramme. Ce n'est pas une bonne méthode si la taille mémoire est limitée, car les tableaux employés sont très importants :

```
10 MODE 1

20 GOSUB 1000

30 GOSUB 2000

40 GOSUB 3000

50 END

999 REM on utilise des entiers
```

```
1000 DEFINT a,c,n,r,v

1005 READ centrex,centrey

1010 READ radius

1020 READ numberofvalues

1030 DIM value(numberofvalues),angle(num
berofvalues)

1040 totalofvalues=0

1050 FOR count=1 TO numberofvalues

1060 READ value(count)

1070 totalofvalues=totalofvalues+value(c
ount)

1080 NEXT

1090 RETURN

1100 DATA 200,200,120,4,1,2,3,4

1998 REM

2000 radval=radius*10

2001 totangle=0

2005 FOR count=1 TO numberofvalues

2010 angle(count)=radval*value(count)/to
talofvalues

2015 totangle=totangle+angle(count)

2020 NEXT

2030 RETURN

3000 ORIGIN centrex,centrey

3010 DEFINT c

3020 count(angle)=0

3030 colour=1
```

```
3033 REM pour calculer un sinus ou un co
sinus plus vite

3035 thesin=SIN(2*PI/radval):thecos=COS(
2*PI/radval)

3036 x1=radius:y1=0

3037 REM calcul des coordonnees des poin
ts formant la circonference

3038 GOSUB 4000

3039 REM mode 0 change la couleur des po
ints (15)

3040 noofcolours=3

3050 FOR count=1 TO numberofvalues

3069 REM on change la couleur des secteu
rs

3070 colour=1+(colour+1)MOD noofcolours

3079 REM

3080 IF count=numberofvalues AND numbero
fvalues MOD noofcolours=1 THEN colour=1+
(colour+1)MOD noofcolours

3099 REM dessin des secteurs

3100 FOR count1=countangle TO countangle
+angle(count)

3114 MOVE 0,0:DRAW x(count1),y(count1),c
olour

3120 NEXT

3124 REM mise a jour de la position suiv
ante

3125 countangle=countangle+angle(count)

3140 NEXT

3150 RETURN

3997 REM
```

```
4000 DIM x(totangle),y(totangle)
4010 FOR count=1 TO totangle
4020 x=x1*thecos-y1*thesin
4030 y=x1*thesin+y1*thecos
4040 x(count)=x
4050 y(count)=y
4060 x1=x: y1=y
4070 NEXT
4080 RETURN
```

Les programmes ci-dessus montrent qu'il est difficile de s'assurer qu'une zone est totalement colorée. La certitude d'une coloration complète ne peut être obtenue qu'en traçant les points un par un, comme nous le verrons dans le prochain chapitre.

EXERCICES

1. Modifier le programme de tracé d'un diagramme sectoriel pour qu'un label soit attribué à chaque secteur.
2. Modifier le programme pour que plusieurs diagrammes de même rayon soient dessinés sur l'écran (c'est l'occasion d'employer des tableaux qui évitent de recalculer les points de la circonférence pour chaque cercle).
3. Écrire un programme qui superpose les uns au-dessus des autres des petits diagrammes sectoriels avec des secteurs colorés.

4

MODÈLES ET IMAGES

Nous avons vu dans le Chapitre 1 qu'il est facile de générer des modèles à l'aide de la combinaison des commandes MOVE et DRAW :

```
10 MODE 1

20 x=320: y=200

30 maximum=200

40 stepsize=5

50 FOR count=0 TO maximum STEP stepsize

60 MOVE x-count, y

70 DRAW x, y+(maximum-count)

80 DRAW x+count, y

90 DRAW x, y-(maximum-count)

100 DRAW x-count, y

110 NEXT
```

Les "mailles de courbes" sont courantes en mathématiques ; de nombreux effets peuvent être générés en reliant simplement une série de points par des lignes droites. Le programme suivant utilise ce principe en traçant d'abord un polygone composé d'un nombre de côtés donné, puis en reliant les sommets entre eux :

```
10 MODE 1

20 radius=150

30 x=320: y=200

40 INPUT"combien de cotes a la figure";s
ides

50 CLS

60 stepsize=2*PI/sides

70 DIM x(sides), y(sides)

80 count=0

90 ORIGIN x, y
```

```
100 MOVE 0,radius

110 FOR angle=0 TO 2*PI STEP stepsize

120 DRAW radius*SIN(angle),radius*COS(an
gle)

130 x(count)=radius*SIN(angle):y(count)=
radius*COS(angle)

140 count=count+1

150 NEXT

155 DRAW 0,radius

160 FOR count1=1 TO sides-1

170 FOR count2=count1+1 TO sides

180 MOVE x(count1),y(count1)

190 DRAW x(count2),y(count2)

200 NEXT

210 NEXT
```

Le résultat est beaucoup plus impressionnant si on ajoute de la couleur :

```
155 colour=1:DRAW 0,radius,colour

160 FOR count1=1 TO sides-1

170 FOR count2=count1+1 TO sides

173 REM differentes couleurs

175 colour=1+(colour+1) MOD 3

180 MOVE x(count1),y(count1)

190 DRAW x(count2),y(count2),colour

200 NEXT

210 NEXT
```

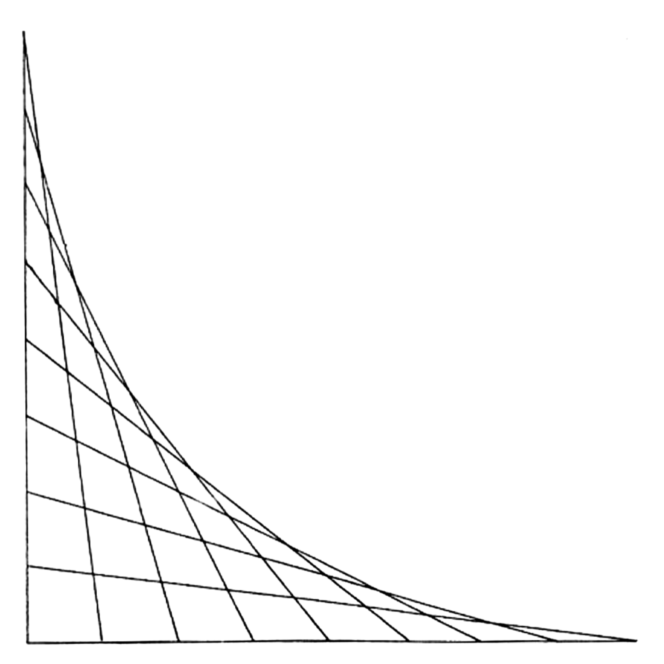


Figure 4.1 : Exemple de mailles de courbes.

Dans ce chapitre, nous allons voir que de nombreux modèles peuvent être générés. La plupart d'entre eux sont basés sur l'utilisation des fonctions sinus et cosinus. Il ne faut pas se laisser impressionner, les programmes suivants sont complets, même s'il permettent à l'utilisateur de tester différents effets en substituant ses propres valeurs à celles des variables.

MODÈLES MOIRÉS

L'utilisation de la commande ORIGIN et des déplacements relatifs facilite le dessin de modèles symétriques. Ce programme utilise comme origine le centre de la zone de dessin. On crée le modèle en reliant des points du nouvel axe des x à des points placés en haut et en bas de l'écran. Chaque coordonnée x est multipliée par un facteur pour que les lignes convergent ou divergent :

```
10 MODE 1

19 REM on utilise des entiers pour plus
de rapidite
```

```
20 DEFINT c,f,x,y

30 xorigin=320:yorigin=200

40 factor1=3:factor2=1

50 factord=factor1-factor2

60 ORIGIN xorigin,yorigin

69 REM la boucle trace 4 lignes symetriq
ues

70 FOR count=0 TO 200

80 MOVE 0,0:MOVER count*factor1,0

90 DRAWR -count*factord,200

100 MOVER -count*factor2*2,0

110 DRAWR -count*factord,-200

120 DRAWR count*factord,-200

130 MOVER count*factor2*2,0

140 DRAWR count*factord,200

150 NEXT
```

On remarque la commande DEFINT de la ligne 20 qui permet d'accélérer le déroulement du programme. Il ne sera pas toujours possible de définir toutes les variables comme des entiers car de nombreux modèles de cette section sont générés à l'aide de fonctions trigonométriques qui donnent des valeurs décimales.

Les facteurs de la ligne 40 peuvent avoir n'importe quelle valeur inférieure à 15 (sinon le modèle est difficile à discerner). Les modèles moirés qui apparaissent sur l'écran résultent du fait que l'Amstrad laisse certains pixels dans la couleur du fond et du chevauchement partiel des lignes. Il faut essayer différents facteurs et faire tourner le programme dans d'autres modes. On peut obtenir un très joli "tapis tissé" en ajoutant ces lignes :

```
35 colour1=1:colour2=2

70 FOR count=0 TO 200
```

```
80 MOVE 0,0: MOVER count*factor1,0

90 DRAWR -count*factord, 200, colour1

100 MOVER -count*factor2*2, 0

110 DRAWR -count*factord, -200, colour2

120 DRAWR count*factord, -200, colour1

130 MOVER count*factor2*2, 0

140 DRAWR count*factord, 200, colour2

150 colour1=1+(colour1+1)MOD 4: colour2=1
+(colour2+1)MOD 4

160 NEXT
```

La ligne 150 génère une couleur de premier plan compatible avec le mode 1. La commande MOD donne le reste de la division : par exemple, 5 MOD 4 est égal à 1. On ajoute 1 au résultat pour éviter d'obtenir la couleur du fond ; sinon 4 MOD 4, qui a pour valeur 0, tracerait un modèle composé de lignes confondues avec le fond. Dans cet exemple, les lignes 35 et 150 se combinent pour afficher un tapis orange. La couleur PEN utilisée avec les commandes DRAWR est toujours 1 (jaune) ou 3 (rouge). La proximité des lignes (qui dépend des facteurs de la ligne 40) détermine la couleur du tapis, soit orange, soit composé de bandes rouges et jaunes.

En attribuant les valeurs 2 et 3 aux variables couleur 1 et 2 (en modifiant le modulo de la division, ligne 150), on obtient une variété d'effets : des modèles dont les côtés diagonalement opposés sont de différentes couleurs ou des modèles composés de certaines couleurs. Le déroulement du programme dans d'autres modes donne des résultats surprenants, sauf si on ajuste la gamme des couleurs à l'aide des instructions de la ligne 35.

FIGURES DE LISSAJOUS

Les figures de Lissajous sont créées à partir de la méthode utilisée pour le tracé d'un cercle. On garde un rayon constant et on prend le sinus et le cosinus du même angle pour déterminer un point de

la circonférence. On obtient ensuite de nombreux modèles en faisant varier l'angle choisi :

```
10 MODE 1

20 xorigin=320: yorigin=200

30 ORIGIN xorigin, yorigin

40 MOVER 100, 0

50 FOR angle=0 TO 32 STEP PI/30

60 DRAW 100*COS(angle),100*SIN(angle*0.8
)

70 NEXT
```

Une autre méthode consiste à calculer les points sur deux courbes et à les relier avec des lignes droites :

```
10 MODE 1

20 xorigin=320: yorigin=200

30 ORIGIN xorigin, yorigin

40 MOVER 100, 0

50 FOR angle=0 TO 6.4 STEP PI/35

55 MOVE 200*SIN(angle),100*COS(angle)

60 DRAW 100*COS(angle),200*SIN(angle)

70 NEXT
```

Voici un autre exemple :

```
10 MODE 1

20 xorigin=320: yorigin=200

30 ORIGIN xorigin, yorigin

40 MOVER 100, 0

50 FOR angle=0 TO 6.4 STEP PI/35
```

```
55 MOVE 300*SIN(angle),50*COS(angle)
60 DRAW 10*COS(angle/5),200*SIN(angle*2)
70 NEXT
```

Nous allons ajouter de la couleur :

```
10 MODE 1
20 xorigin=320: yorigin=200
30 ORIGIN xorigin, yorigin
40 colour=1
50 FOR angle=0 TO 20 STEP PI/30
53 IF angle>10 THEN colour=3
55 MOVE 200*SIN(angle),200*COS(angle)
60 DRAW 100*COS(angle*3),200*SIN(angle/3
),colour
70 NEXT
```

Comme pour les programmes précédents, il faut essayer de les faire tourner dans un autre mode. On peut croire que les fonctions sinus et cosinus qui génèrent les modèles fournissent des résultats imprévisibles ; cependant, on s'aperçoit vite de l'effet obtenu par la modification des variables. Il est possible de faire des essais avec des fonctions plus complexes, comme par exemple le carré du sinus ou du cosinus, ou bien leur multiplication/division par d'autres facteurs.

SPIRALES

Les spirales sont générées par le programme de tracé de cercle modifié de telle sorte que le rayon ne soit pas constant :

```
10 MODE 1
20 GOSUB 1000
100 END
```

```
1000 xorigin=315: yorigin=190

1010 ORIGIN xorigin, yorigin

1020 colour=1

1030 increaseradius=0.5

1040 stepsize=PI/30

1048 REM endangle determine le nombre de
 spirales

1050 endangle=40

1060 startradius=1

1070 GOSUB 2000

1900 RETURN

2000 MOVE 0,0

2010 FOR angle=0 TO endangle STEP stepsi
ze

2020 DRAW startradius*SIN(angle),startra
dius*COS(angle),colour

2030 startradius=startradius+increaserad
ius

2040 NEXT

2050 RETURN
```

L'addition de quelques lignes permet de tracer plusieurs spirales imbriquées :

```
1080 startradius=10

1090 GOSUB 2000

1100 startradius=20

1110 GOSUB 2000
```

La spirale peut être animée de manière à simuler une rotation ; il suffit de tracer les points en différentes couleurs puis d'utiliser la com-

mande INK pour afficher alternativement la couleur du fond et celle du dessin :

```
10 MODE 1

20 GOSUB 1000

29 REM initialisation des INK pour que l
es points changent de couleur

30 INK 1,1,20

40 INK 2,20,21

50 reponse$=""

60 WHILE reponse$=""

70 reponse$=INKEY$

80 WEND

90 INK 1,24:INK 2,20

100 END

1054 REM ink 1=ink 2

1055 INK 1,20

2015 IF colour=1 THEN colour=2 ELSE colo
ur=1
```

C'est une technique très utile que nous utiliserons encore au cours de cet ouvrage.

MODÈLES RÉPÉTITIFS

De nombreux modèles de papier peint sont générés par la répétition d'un motif :

```
10 MODE 1

20 GOSUB 1000

40 END

998 REM donnees pour tracer un octogone
```

```
1000 xstart=0: ystart=0

1001 REM xstart et ystart determinent le
 centre du premier polygone

1010 radius=40

1020 sides=8

1030 stepsize=2*PI/sides

1040 colour=2

1050 GOSUB 3000

1060 RETURN

3000 centrex=xstart: centrey=ystart

3010 WHILE centrex<639 OR centrey>399

3020 ORIGIN centrex, centrey

3030 MOVE 0, radius

3040 FOR angle=0 TO 2*PI STEP stepsize

3050 DRAW radius*SIN( angle), radius*COS( a
ngle), colour

3060 NEXT

3070 centrex=centrex+radius*2

3079 REM si hors ecran on recommence

3080 IF centrex>639 AND centrey<399 THEN
 centrex=xstart: centrey=centrey+radius*2

3090 WEND

3100 RETURN
```

Des résultats plus élaborés sont obtenus si un second jeu de figures est superposé au premier (ces figures peuvent être complètement différentes ou bien être des versions agrandies ou diminuées des premières) :

```
30 GOSUB 2000

1999 REM donnees pour tracer un hexagone
```

```
2000 xstart=40: ystart=40

2010 radius=40

2020 sides=6

2030 stepsize=2^PI/sides

2040 colour=2

2050 GOSUB 3000

2060 RETURN
```

Les rangées sont animées si leur origine est déplacée :

```
3079 REM si hors ecran on recommence

3080 IF centrex>639 AND centrey<399 THEN
 GOSUB 4000

3090 WEND

3100 RETURN

4000 IF xstart=0 THEN xstart=radius ELSE
 xstart=0

4010 centrex=xstart: centrey=centrey+radi
us^2

4020 RETURN
```

ROTATION DE FORMES

Il est relativement simple de modifier le programme de tracé de cercle pour qu'il dessine un polygone quelconque.

Les lignes suivantes peuvent être incorporées sous la forme d'un sous-programme dans un programme qui agrandit et fait tourner une forme donnée en créant un modèle de spirale :

```
10 MODE 1

20 GOSUB 1000

30 GOSUB 1500

40 END
```

```
999 REM donnees pour tracer le polygone

1000 READ sides

1010 READ radius

1020 READ centrex,centrey

1030 READ radiuschange,anglechange

1040 colour=2

1050 stepsize=2*PI/sides

1060 startangle=0:finishangle=2*PI

1070 RETURN

1500 ORIGIN centrex,centrey

1510 WHILE radius<200

1520 MOVE radius*SIN(startangle),radius*
COS(startangle)

1530 FOR angle=startangle TO finishangle
 STEP stepsize

1540 DRAW radius*SIN(angle),radius*COS(a
ngle),colour

1550 NEXT

1560 DRAW radius*SIN(startangle),radius*
COS(startangle)

1569 REM on incremente radius

1570 radius=radius+radiuschange

1580 startangle=startangle+anglechange

1590 finishangle=finishangle+anglechange

1600 WEND

1610 RETURN

2000 DATA 3,20,300,200,5,1

2010 DATA 3,20,300,200,3,10
```

```
2020 DATA 4,30,300,200,4,3
2030 DATA 6,20,300,200,1,6
2040 DATA 6,20,300,200,6,1
2050 DATA 8,10,300,200,5,10
2060 DATA 8,10,300,200,5,2
```

On fait tourner le programme. Les lignes DATA commençant ligne 2000 contiennent les données relatives à un certain nombre de figures. On efface la première ligne DATA et on fait tourner à nouveau le programme pour voir quel est l'effet de la variation du nombre de côtés et à quelle vitesse la figure est agrandie et déplacée. Cette opération est répétée pour les autres lignes de données.

Certains effets très impressionnants sont obtenus grâce à l'emploi judicieux de la couleur :

```
10 MODE 1
20 GOSUB 1000
29 REM INK 2 et 3 pour les points
30 GOSUB 1500
40 INK 2,1,20
50 INK 3,20,1
59 REM on attend la frappe d'une touche
60 reponse$=""
70 WHILE reponse$=""
80 reponse$=INKEY$
90 WEND
99 REM retour a la normale
100 INK 2,20
```

```
110 INK 3,6

120 END

1504 REM INK3=INK2

1505 INK 3,20

1595 IF colour=2 THEN colour=3 ELSE colo
ur=2
```

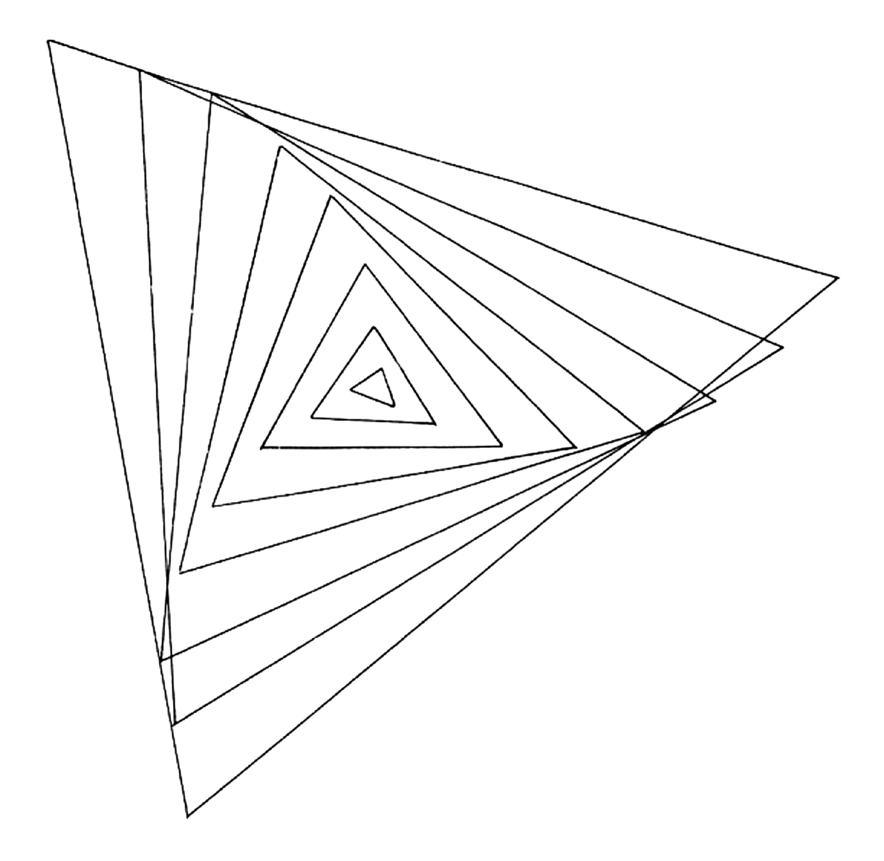


Figure 4.2 : Type de modèle généré par la rotation d'un modèle de base.

La frappe d'une touche quelconque après le dessin de la figure produit un effet irréel engendré par l'alternance des couleurs du fond et du dessin.

EXERCICES

1. Créer une version modifiée du programme de dessin du polygone en lui faisant tracer deux polygones dont les sommets devront être reliés.
2. Tester les commandes suivantes sur le programme qui génère un modèle moiré : modifier le pas de la boucle FOR ... NEXT ; superposer un second modèle en utilisant des facteurs et des couleurs différentes ; déplacer l'origine pour créer un *patchwork*.
3. Modifier le programme de rotation du polygone pour que le nombre de côtés du polygone varie au cours du déroulement. On peut alterner par exemple un triangle et un pentagone ou ajouter un côté au polygone à chaque rotation ; le cycle reprend lorsque la figure comporte 20 côtés.

DESSIN SUR L'ÉCRAN

Au lieu de laisser l'ordinateur créer son propre modèle, on peut écrire un programme qui permet à l'utilisateur de dessiner et de manipuler des formes sur l'écran. Nous allons étudier cette possibilité dans le reste de ce chapitre. Toutes les commandes sont exécutées à partir du clavier pour que ces programmes soient accessibles à un public le plus large possible, mais ils peuvent être modifiés de façon à être utilisés avec des manettes de jeu.

Au cours de cette section, nous travaillerons en mode 0, celui qui offre la gamme de couleurs la plus importante. Le premier module contient les caractéristiques essentielles de n'importe quel programme de dessin : il permet de tracer des points et des lignes dans différentes directions.

```
10 MODE 0
20 x=320: y=200
30 colour=1
40 MOVE x,y
50 PLOT x,y,colour
60 GOSUB 1000
```

```
70 END

997 REM on scrute le clavier en attendan
t la frappe de 'e'

1000 WHILE reponse$<>"e"

1010 reponse$=INKEY$

1020 IF reponse$="a" THEN y=y+2

1030 IF reponse$="z" THEN y=y-2

1040 IF reponse$="," THEN x=x-4

1050 IF reponse$="." THEN x=x+4

1060 PLOT x,y,colour

1070 WEND

1080 RETURN
```

Tout d'abord, un point est tracé aux coordonnées spécifiées ligne 40. La commande INKEY$ de la ligne 60 vérifie si une touche du clavier a été tapée ; le déplacement est généré par la frappe des touches a/z (déplacement vertical) ou ,/. (déplacement horizontal). Les coordonnées du point sont mises à jour et la position du nouveau point est tracée.

On remarque que la structure de ce petit programme rend l'adjonction de commandes supplémentaires très facile. Pour l'instant, le déplacement du point est uniquement vertical ou horizontal, mais un déplacement en diagonale est possible si l'on ajoute des lignes à l'intérieur de la boucle des lignes 1000 à 1080.

Sous cette forme, le programme n'autorise que des lignes continues et empêche tout déplacement sur une nouvelle position sans tracé. On peut ajouter des options pour que le point soit tracé soit dans la couleur du fond, soit dans la couleur du dessin lors de la frappe respective de "f" ou de "b" :

```
10 IF reponse$ = "f" THEN couleur = 1
20 IF reponse$ = "b" THEN couleur = 0
```

Malheureusement, une fois la couleur du fond sélectionnée, le point est invisible, ce qui rend son déplacement difficile. On peut contourner cette difficulté en traçant le point deux fois :

```
35 couleurdessin = 1
1005 PLOT x,y,couleur
1060 PLOT x,y,couleurdessin
```

Si la couleur du fond est sélectionnée, le point placé à la position x,y est tracé de façon invisible (ligne 1005) puis tracé dans la couleur du dessin (ligne 1060). En conséquence, le point clignote et sert de curseur pour identifier la position en cours.

On peut effectuer un changement de couleur du dessin, soit en sélectionnant une couleur par la frappe d'une touche particulière, soit en ajoutant une ligne de programme :

```
1053 IF reponse$ = "c" THEN couleur = 1
+ (couleur + 1) MODE 3
```

Cela permet un choix de trois couleurs qui génèrent des lignes différentes.

Il est parfois difficile de voir la couleur du point ; par exemple, le tracé précis d'un rectangle n'est pas évident à l'œil nu. On pourrait se contenter d'afficher cette information sur l'écran mais nous allons profiter de cette occasion pour introduire la commande WINDOW qui définit sur l'écran une zone réservée à l'affichage de texte :

```
15 WINDOW 1,40,24,25
16 PRINT "Couleur : "
17 PRINT "x ;        y :    ";
```

Les quatre nombres suivant la commande WINDOW spécifient respectivement les coordonnées texte x définissant les côtés gauche et droit de la fenêtre, puis les coordonnées texte y définissant ses côtés inférieur et supérieur. Dans cet exemple, le texte sera affiché sur les lignes 24 et 25 :

```
1051 IF reponse$="f" THEN colour=foregro
undcolour

1052 IF reponse$="b" THEN foregroundcolo
ur=colour: colour=0

1053 IF reponse$="c" THEN colour=1+(colo
ur+1) MOD 3

1060 PLOT x,y,foregroundcolour

1065 GOSUB 2000

1070 WEND

1080 RETURN

2000 IF oldcolour<>colour THEN LOCATE 9,
1: PRINT colour

2010 IF oldx<>x THEN LOCATE 4,2: PRINT x;

2020 IF oldy<>y THEN LOCATE 12,2: PRINT y
;

2030 oldcolour=colour

2040 oldx=x: oldy=y

2050 RETURN
```

Les lignes peuvent maintenant être placées sur l'écran avec précision car le programme fournit une mise à jour permanente de la couleur PEN du point en cours et de ses coordonnées x et y. On remarque que l'Amstrad considère toujours la fenêtre texte comme faisant partie intégrante de l'écran graphique. On s'aperçoit qu'il est possible de superposer des points au texte.

Le programme comporte toujours un certain nombre d'imperfections. Les lignes ne peuvent être effacées qu'en étant retracées dans la couleur du fond. Cela entraîne un autre problème quand le point établi dans la couleur du fond croise une ligne déjà dessinée ; dans ce cas, une partie de la ligne est effacée. Ce problème est cependant facile à résoudre si nous faisons pour cela appel à nos connaissances en binaire.

EXERCICES

1. Le programme de dessin ne permet pas de vérifier si le point tracé se trouve en dehors de l'écran. Modifier ce programme pour qu'il soit impossible de sortir de l'écran ou de dessiner dans la fenêtre texte.
2. Étendre la gamme des couleurs pour qu'elles soient au nombre de seize. Utiliser la commande INK pour que les couleurs affichées ne soient pas seulement les couleurs par défaut du mode 0 mais soient initialement choisies à partir des vingt-sept couleurs disponibles.
3. Ajouter quelques commandes supplémentaires pour que des lignes diagonales puissent être dessinées.

UTILISATION DE EXOR

Dans un chapitre précédent, nous avons étudié la correspondance entre les nombres binaires et les nombres hexadécimaux ; nous allons voir maintenant l'intérêt de ces nombres pour l'exécution d'opérations graphiques.

Il ne faut pas oublier que l'écran d'affichage est en fait la représentation d'une partie de la mémoire de l'ordinateur. Tous les caractères affichés et chaque ligne graphique dessinée sont générés par des valeurs binaires particulières stockées dans des zones mémoire que l'Amstrad examine pour élaborer son écran d'affichage.

Toute ligne dessinée sur l'écran entraîne la modification des octets (valeurs binaires sur huit bits) correspondants dans la mémoire graphique. La valeur de ces octets détermine si un pixel doit être allumé ou éteint et, s'il est allumé, quelle est sa couleur. En fait, les couleurs de chaque pixel sont dérivées d'un octet unique d'une façon assez complexe que nous n'étudierons pas ici. Pour le problème qui nous concerne, nous allons utiliser un modèle simplifié de relation octet-pixel et considérer que la valeur d'un octet unique stocké en mémoire indique à l'Amstrad la valeur d'un seul pixel à afficher sur l'écran.

On suppose que seules quatre couleurs sont utilisées et que les octets représentant la mémoire écran ont donc tous l'une des quatre valeurs de la Figure 4.3.

Code binaire	Couleur
00000000	Bleu
00000001	Blanc
00000010	Jaune
00000011	Rouge

Figure 4.3 : Représentations binaires possibles qui peuvent être utilisées par l'ordinateur pour indiquer quatre couleurs différentes.

Si l'écran était complètement bleu, tous les octets auraient pour valeur 00000000 ; s'il était complètement blanc, ils auraient pour valeur 00000001 etc. Le dessin d'une ligne blanche sur l'écran a pour conséquence la modification des octets des pixels concernés (ils ont la valeur 00000001). Il en est de même pour une ligne jaune avec la valeur 00000010. Nous avons vu précédemment que les codes ASCII les plus bas donnent à l'Amstrad des commandes particulières telles que *Déplacer le curseur d'un caractère en arrière* ou *Éteindre l'écran*. Un de ces codes influence la manière dont l'Amstrad traite les points graphiques.

On peut utiliser le code ASCII 23 pour que l'Amstrad dessine des points sur l'écran à l'aide de l'opérateur logique OU exclusif (EXOR ou XOR). Sans cette option, l'Amstrad remplace simplement l'ancienne valeur de l'octet par la nouvelle. Par exemple, pour une ligne dessinée en jaune, tous les octets de la ligne ont la valeur 00000010. L'utilisation de XOR provoque le tracé de tous les points en combinant l'ancienne valeur de chaque octet avec sa nouvelle valeur en fonction de certaines règles déterminées.

Nous allons examiner un octet sur l'écran ; pour commencer, l'octet a pour valeur 00000000 c'est-à-dire qu'il indique un pixel coloré en bleu, couleur du fond, comme dans la Figure 4.4. Une ligne jaune passe par ce point (l'octet a pour valeur 00000010) comme dans la Figure 4.5.

00000000	Point dans la couleur du fond, bleu.

Figure 4.4

00000000	Point bleu et
00000010	ligne jaune passant par ce point.

Figure 4.5

Étant donné que l'option XOR est établie, l'Amstrad ne remplace pas seulement l'octet 00000000 (bleu) par l'octet 00000010 (jaune) ; il combine la valeur des deux octets. Si les bits correspondants sont différents, le résultat est 1 ; si les bits correspondants sont identiques, le résultat est 0. L'affichage se termine par un point jaune, ce que nous aurions obtenu si nous n'avions pas utilisé l'option XOR. Cependant, si on trace maintenant une ligne jaune identique sur la ligne qui vient d'être dessinée, l'effet est assez inattendu (voir la Figure 4.7). Étant donné que les bits constituant l'ancien octet et ceux qui constituent le nouvel octet sont parfaitement identiques, l'option EXOR génère un octet qui a pour valeur 00000000. La ligne dessinée en bleu, couleur du fond, disparaît donc. Le dessin d'une autre ligne jaune nous replace dans la situation de la Figure 4.4 et la ligne réapparaît.

00000000	Un point bleu et
EXOR 00000010	une ligne jaune passant par ce point
00000010	donnent un point jaune.

Figure 4.6

00000010	Un point jaune et
EXOR 00000010	une ligne jaune passant par ce point
00000000	donnent un point bleu, couleur du fond.

Figure 4.7

00000001	Un point blanc croisé par
EXOR 00000010	une ligne jaune.

Figure 4.8

Que se passe-t-il si une ligne jaune est croisée par une ligne blanche ? L'option EXOR génère un point rouge comme dans la Figure

4.9. De nouveau, le second dessin d'une ligne de la même couleur restitue le dessin initial comme dans la Figure 4.10.

00000001	Une ligne blanche croisée par
EXOR 00000010	une ligne jaune donne un
00000011	point rouge.

Figure 4.9

00000011	Un point rouge croisé par
EXOR 00000010	une ligne jaune donne
00000001	un point blanc.

Figure 4.10

L'opérateur logique EXOR peut ne pas sembler très utile, mais les effets graphiques obtenus sont inestimables dans n'importe quel programme de dessin. Des lignes peuvent être dessinées et effacées sans que cela affecte les autres lignes ; on peut faire des essais et déplacer les lignes sur des positions différentes avant de leur attribuer une place définitive à l'aide des commandes de dessin normal. La seule chose à faire est de s'assurer que toute ligne (ou point) est tracée deux fois par la commande EXOR de telle sorte qu'elle disparaisse.

DESSIN DE DIAGRAMMES

Avant d'examiner un programme de dessin utilisant l'option EXOR, nous allons résumer les caractéristiques qui peuvent être rajoutées au dessin :

1. Lignes fixées de manière temporaire ou permanente.
2. Effacement de lignes.
3. Sauvegarde de dessins.
4. Dessin de formes standard telles que cercles, rectangles etc.
5. Translation, agrandissement ou autre transformation d'une partie du dessin.

Les deux derniers points de cette liste seront traités dans un prochain chapitre. Nous venons de découvrir comment dessiner des

lignes temporaires ; par conséquent, le point 1 ne présente aucun problème. Le moyen le plus simple pour parvenir à exécuter les points 2 et 3 consiste à sauvegarder les coordonnées définitives dans un tableau. Cela permet d'identifier et d'effacer une ligne facilement et rend possible la sauvegarde du dessin dans un fichier. Il suffit de sauvegarder la liste des coordonnées stockées dans un tableau et de les utiliser pour reproduire le dessin ultérieurement.

Nous allons utiliser une structure modulaire pour que le programme puisse être étendu sans problème :

```
10 MODE 0
20 DIM x(100),y(100)
30 startx=320: starty=200
40 x=320: y=200
50 foregroundcolour=1
60 PRINT CHR$(23);CHR$(1);
70 GOSUB 1000
80 GOSUB 2000
90 END
999 REM trace de ligne
1000 PLOT x,y,foregroundcolour
1010 DRAW startx,starty
1020 RETURN
2000 WHILE reponse$<>"e"
2010 GOSUB 1000
2020 reponse$=LOWER$(INKEY$)
2030 IF reponse$="a" THEN y=y+2
2040 IF reponse$="z" THEN y=y-2
2050 IF reponse$="," THEN x=x-4
```

```
2060 IF reponse$="." THEN x=x+4
2070 IF reponse$=" " THEN GOSUB 3000
2080 GOSUB 1000
2090 WEND
2100 RETURN
3000 PRINT CHR$(23);CHR$(0);
3010 GOSUB 1000
3020 PRINT CHR$(23);CHR$(1);
3030 count=count+1
3040 x(count)=x: y(count)=y
3050 startx=x: starty=y
3060 RETURN
```

La ligne 20 établit deux tableaux qui manipulent les coordonnées de 100 points (ce nombre est arbitraire et peut être augmenté). Les coordonnées du point en cours sont données par les valeurs des variables x et y. Les valeurs de debx et deby donnent les coordonnées du dernier point "fixé". Ces coordonnées doivent être disponibles pour que l'on puisse tracer une ligne permanente entre les deux points si nécessaire.

Le sous-programme de la ligne 2000 est le module conducteur du programme ; il scrute le clavier pour recevoir les entrées des données et trace ou efface successivement une ligne définie par les points debx,deby et x,y. Si on déplace le point x,y, on voit dans le sous-programme de la ligne 1000 que l'Amstrad dessine une ligne jaune vacillante entre ce point et le point debx,deby.

La frappe de la barre d'espacement fixe la ligne de façon permanente par l'intermédiaire du sous-programme 3000. La ligne 3000 replace le mode graphique, trace définitivement la ligne, puis replace l'ordinateur en option EXOR pour que le tracé continue. Les coordonnées x et y du point en cours sont stockées dans les tableaux x() et y() et de nouvelles valeurs sont attribuées à debx et deby pour que la ligne suivante soit dessinée.

On peut modifier le programme pour dessiner ou non une ligne :

```
55 linedraw=0

999 REM trace d'une ligne (non si linedr
aw=0)

1000 PLOT x,y,foregroundcolour

1005 IF linedraw=0 THEN RETURN

1010 DRAW startx,starty

1020 RETURN

2070 IF reponse$=" " THEN GOSUB 3000:lin
edraw=1

2071 IF reponse$="1" THEN IF linedraw=0
THEN linedraw=1 ELSE linedraw=0
```

La ligne 2071 utilise la touche ''l'' comme commutateur pour tracer ou non une ligne. Quand tralign = 0, la ligne n'est pas dessinée, puisque le sous-programme de la ligne 1000 se termine avant que le point (x,y) soit relié au point (debx,deby). Le déroulement du programme montre qu'il n'est plus nécessaire de dessiner la première ligne à partir du point (debx,deby) et que le point peut être déplacé n'importe où avant d'être fixé par la frappe de la touche ''f''.

L'effacement d'une ligne est un peu délicat ; seul l'effacement de la dernière ligne tracée est autorisé, il est exécuté par l'intermédiaire de la frappe de la touche ''d''.

```
2072 IF reponse$="d" THEN GOSUB 4000

3998 REM ne fonctionne pas si on essaie
d'effacer une ligne qui n'existe pas

4000 x=x(count):y=y(count)

4010 count=count-1

4020 startx=x(count):starty=y(count)

4030 GOSUB 1000

4040 RETURN
```

Cela permet l'effacement d'une ligne quelconque ainsi que celui des lignes tracées ultérieurement.

Si la réponse du clavier est un peu lente, on peut ajouter les lignes suivantes :

```
1 DEFINT c,f,l,o,s,x,y
2 SPEED KEY 2,2
89 SPEED KEY 10,5
```

Nous avons déjà remarqué que l'utilisation de nombres entiers accélère le déroulement d'un programme. La commande SPEED KEY est suivie de deux nombres qui représentent le retard de la répétition des touches et la période de répétition ; l'unité est 1/50 de seconde. Ces deux valeurs déterminent la rapidité de réponse de l'Amstrad à la frappe d'une touche. Quand une touche est tapée, l'ordinateur attend un certain temps égal au retard de répétition avant de répéter les caractères. A partir de ce moment, le caractère est répété à certains intervalles gérés par la période de répétition. Il est essentiel de redonner à la commande SPEED KEY ses valeurs par défaut à la fin du programme. Une réponse trop rapide à la frappe des touches peut rendre l'écriture d'une instruction cohérente virtuellement impossible. Pour pouvoir sauvegarder un dessin, il est nécessaire de modifier légèrement le programme. L'enregistrement des coordonnées de tous les points n'est plus adapté et il suffit de savoir si un point est relié au point précédent ou non :

```
20 DIM x(100),y(100),l(100)

3030 count=count+1

3040 x(count)=x: y(count)=y

3045 l(count)=linedraw

3050 startx=x: starty=y

3060 RETURN

4000 x=x(count): y=y(count): linedraw=l(co
unt)
```

Les tableaux x() et y() manipulent les coordonnées de tous les points. Un troisième tableau 1() est introduit pour indiquer si un point est

relié au précédent. Chaque fois qu'un point est fixé, 1(compt) est utilisé pour enregistrer la condition en cours de *tralign*. Si celle-ci est égale à 0, la ligne n'est pas dessinée et le point en cours n'est pas relié au précédent. Toute valeur différente de 0 montre que la ligne a été tracée et que le point est relié au précédent.

```
2074 IF reponse$="i" THEN GOSUB 7000

2075 IF reponse$="o" THEN GOSUB 8000

6000 CLG

6010 startx=x(1):starty=y(1)

6020 FOR value=2 TO count

6030 x=x(value):y=y(value)

6040 linedraw=l(value)

6050 GOSUB 1000

6060 startx=x:starty=y

6070 NEXT

6080 x=320:y=200:linedraw=0

6090 RETURN

7000 MODE 1

7010 PRINT"pour charger des donnees"

7020 INPUT"donnez le nom du fichier";fil
e$

7030 OPENIN file$

7040 count=0

7050 WHILE NOT EOF

7060 count=count+1

7070 INPUT #9,x(count),y(count),l(count)

7080 WEND

7090 CLOSEIN
```

```
7100 MODE 0

7110 WINDOW 1,20,25,25

7120 startx=x(count):starty=y(count)

7130 x=startx:y=starty

7140 GOSUB 6000

7150 RETURN

8000 MODE 1

8010 PRINT"pour sauvegarder une figure"

8020 INPUT"donnez le nom du fichier",fil
e$

8030 openoutfile$

8040 counter=0

8050 WHILE counter<=count

8060 WRITE #9,x(counter),y(counter),l(co
unter)

8070 counter=counter+1

8080 WEND

8090 CLOSEOUT

8100 MODE 0

8110 WINDOW 1,20,25,25

8120 GOSUB 6000

8130 RETURN
```

Le sous-programme de la ligne 8000 écrit toutes les coordonnées et les données relatives à la connexion des points provenant des trois tableaux dans un fichier. Puis on utilise le sous-programme de la ligne

6000 pour recréer le dessin. Après avoir sauvegardé les données, il faut pouvoir les charger ; cela est exécuté par le sous-programme de la ligne 7000. Il n'est pas nécessaire de se placer en mode 1 pour l'exécution de ces sous-programmes, mais, dans ce mode, les messages sont plus faciles à lire. Le sous-programme de chargement est appelé par la frappe de la touche "i" et le sous-programme de sauvegarde est appelé par la frappe de la touche "o" ; ces touches peuvent être remplacées par d'autres touches choisies par l'utilisateur.

Le programme de dessin de base est presque complet. Nous allons simplement ajouter une option couleur :

```
2073 IF reponse$="s" THEN GOSUB 5000

5000 WINDOW 1,20,25,25

5010 INPUT"facteur de modification",scal
e

5020 FOR value=1 TO count

5030 x(value)=scale*(x(value)-x)+320

5040 y(value)=scale*(y(value)-y)+200

5050 NEXT

5060 GOSUB 6000

5070 RETURN
```

L'adjonction de la couleur soulève un problème ; en effet, pour être effacée, une ligne doit être dessinée avec EXOR et dans la couleur correcte. Sinon, une ligne blanche dessinée sur une ligne jaune n'efface pas la ligne initiale mais se contente de modifier sa couleur. Heureusement, il n'est pas nécessaire d'établir un autre tableau pour la couleur car cette information stockée dans 1() peut être utilisée pour l'effacement de ligne ou quand une image est dessinée à l'aide de données chargées à partir d'un fichier.

Nous allons conclure ce chapitre par une démonstration de la souplesse du programme. L'adjonction du sous-programme suivant permet de créer un dessin, de l'agrandir ou de le réduire :

```
2070 IF reponse$=" " THEN GOSUB 3000:lin
edraw=foregroundcolour

2071 IF reponse$="l" THEN IF linedraw=0
THEN linedraw=foregroundcolour ELSE line
draw=0

2076 IF reponse$="c" THEN foregroundcolo
ur=1+(foregroundcolour+1) MOD 3

3045 IF linedraw>0 THEN l(count)=foregro
undcolour

4025 IF linedraw>0 THEN foregrouncolour=
linedraw
```

La frappe de la touche ''s'' appelle le sous-programme de la ligne 5000, qui demande le facteur d'échelle qui servira à réduire ou à agrandir le dessin. Par exemple, la frappe de ''2'' génère un dessin dont la taille est multipliée par deux. De même, ''0.1'' réduit le dessin à 1/10 de sa taille normale.

Le programme utilise la capacité de l'Amstrad à accepter les coordonnées x et y pour les commandes PLOT, MOVE et DRAW, même quand ces coordonnées se trouvent en dehors des limites de l'écran. L'ordinateur trace ces lignes, mais celles-ci ne sont visibles qu'à l'intérieur de la zone graphique définie par l'écran (abscisse comprise entre 0 et 639 ; ordonnée comprise entre 0 et 399).

Une fois le facteur d'échelle choisi, l'Amstrad multiplie toutes les coordonnées des tableaux x() et y() par ce facteur. La position en cours du curseur est utilisée comme point de départ de l'agrandissement. Le nouveau dessin est centré autour du point 320,200 qui est également considéré comme la nouvelle position du curseur.

L'utilisation d'entiers peut sembler une bonne méthode d'accélération du programme, mais elle présente ici un inconvénient, puisqu'elle limite le choix des facteurs d'échelle à des nombres entiers.

EXERCICES

1. Ajouter quelques lignes à la fin du programme de dessin pour restituer le temps de réponse à la frappe des touches du clavier normal.

2. Introduire des processus de vérification pour que le dessin ne puisse pas sortir de l'écran.

3. Ajouter un sous-programme au programme pour permettre à l'utilisateur de sélectionner un temps de retard et une période de répétition des touches de son choix.

4. Ajouter un sous-programme au programme pour que les coordonnées du curseur soient affichées en permanence sur l'écran.

5. Introduire un effacement de lignes sélectif pour que des lignes autres que la dernière ligne créée puissent être effacées. Il sera nécessaire d'établir un cycle pendant lequel toutes les lignes seront effacées puis redessinées à l'aide de la frappe d'une touche jusqu'à ce que la ligne recherchée soit atteinte et définitivement effacée. Il ne faut pas oublier de reporter cet effacement de ligne dans les données stockées dans les tableaux, sinon la ligne réapparaîtra lors du chargement des valeurs du tableau à partir du fichier.

5

ANIMATION ...

Dans le chapitre précédent, nous avons introduit l'opérateur logique EXOR et nous avons vu comment l'utiliser pour le dessin et l'effacement des lignes. Nous allons maintenant l'employer avec d'autres commandes pour améliorer la qualité de l'animation. Une méthode d'animation consiste à dessiner et effacer successivement une image de l'écran. C'est le moyen le plus simple et, si la figure n'est pas trop complexe, la rapidité de l'ordinateur est convenable. Le programme suivant déplace un rectangle sur l'écran en le dessinant et en l'effaçant sur chaque position :

```
10 MODE 1
20 x=100: y=100
30 xdistance=50: ydistance=100
40 xinc=4
50 WHILE x<639
59 REM trace du rectangle
60 colour=1: GOSUB 1000
69 REM effacement du rectangle
70 colour=0: GOSUB 1000
80 x=x+xinc
90 WEND
100 END
999 REM instructions de tracage
1000 MOVE x, y
1010 DRAWR xdistance, 0, colour
1020 DRAWR 0, ydistance
1030 DRAWR -xdistance, 0
1040 DRAWR 0, -ydistance
1050 RETURN
```

Une simple modification déplace le rectangle en diagonale :

```
40 xinc=4 : yinc=2
80 x=x+xinc : y=y+yinc
```

L'adjonction de quelques lignes supplémentaires permet de faire "rebondir" la figure sur l'écran :

```
45 continue=1

50 WHILE continue=1

59 REM trace du rectangle

60 colour=1: GOSUB 1000

69 REM effacement du rectangle

70 colour=0: GOSUB 1000

79 REM mise a jour des coordonnees ( x et
 y)

80 x=x+xinc: y=y+yinc

81 IF x<0 OR x>639 THEN xinc=-xinc: x=x+2
*xinc

82 IF y<0 OR y>399 THEN yinc=-yinc: y=y+2
*yinc

90 WEND
```

Les résultats ne sont pas aussi bons quand de nombreuses lignes sont concernées.

Nous allons maintenant animer un dessin de chien très simplifié (voir la Figure 5.1). Deux figures sont dessinées dans des positions légèrement différentes. L'ordinateur commence par dessiner le chien dans une position, puis il l'efface et le dessine dans l'autre position. Après effacement de la seconde image, les coordonnées sont mises à jour et l'ensemble du cycle est répété. Pour simplifier l'alternance des deux images, il est conseillé de les stocker dans un tableau.

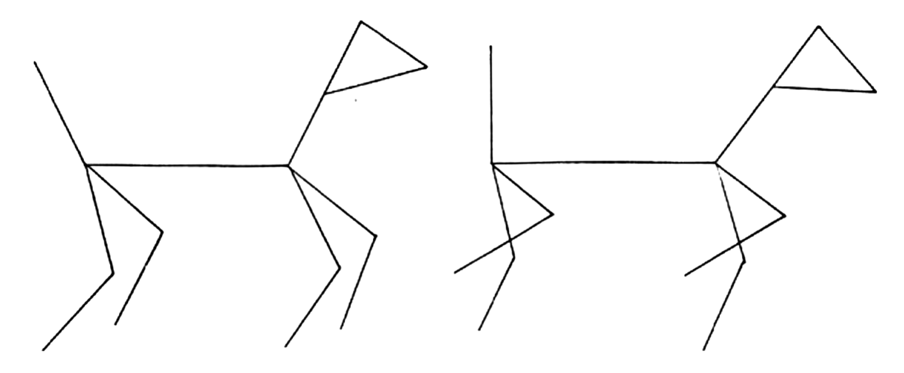


Figure 5.1 : Utilisation de la même figure dans deux positions différentes pour améliorer l'animation.

```
10 MODE 1

20 GOSUB 1000

30 GOSUB 2000

40 END

998 REM on dimensionne 2 tableaux de 26
coordonnees

999 REM pour avoir 2 images du chien

1000 DIM x(100), y(100)

1010 FOR count=1 TO 52

1020 READ x(count), y(count)

1030 NEXT

1040 RETURN

1050 DATA 0, 0, 20, 40, 20, 40, 10, 80, 10, 80, 0,
120, 20, 10, 35, 50, 35, 50, 10, 80, 10, 80, 70, 80

1060 DATA 60, 0, 80, 40, 80, 40, 70, 80, 80, 10, 9
5, 50, 95, 50, 70, 80, 70, 80, 90, 140, 90, 140, 110
, 120, 110, 120, 80, 110

1070 DATA 5, 10, 25, 40, 25, 40, 15, 80, 15, 80, 1
0, 120, 0, 15, 30, 50, 30, 50, 15, 80, 15, 80, 75, 80
```

```
1080 DATA 75,80,85,40,85,40,65,10,75,80,
90,50,90,50,60,15,75,80,100,130,100,130,
120,110,120,110,90,100

1999 REM on dessine et on efface success
ivement le chien

2000 xinc=0

2010 flag=0

2020 WHILE x(1)+xinc<639

2030 IF flag=0 THEN start=1:flag=1 ELSE
start=27:flag=0

2039 REM dessin du chien

2040 colour=1:GOSUB 3000

2049 REM effacement du chien

2050 colour=0:GOSUB 3000

2059 REM changement de position pour des
siner le nouveau chien

2060 xinc=xinc+20

2070 WEND

2080 RETURN

3000 FOR count=start TO start+25 STEP 2

3010 MOVE x(count)+xinc,y(count)

3020 DRAW x(count+1)+xinc,y(count+1),col
our

3030 NEXT

3040 RETURN
```

Dans ce cas, l'ensemble du processus est trop long parce que la figure est effacée et redessinée entièrement chaque fois. Il est donc nécessaire d'utiliser une autre méthode en se servant des possibilités évoquées dans le Chapitre 3 ; l'ordinateur peut modifier la gamme des couleurs disponibles dans un mode grâce à la commande INK.

Si une figure est dessinée à l'aide de la commande PEN initialisée avec la couleur du fond, cette figure peut apparaître instantanément lorsque l'on modifie les paramètres de la commande INK pour qu'elle soit affichée dans la couleur du dessin.

Le programme suivant utilise cette méthode pour afficher alternativement deux rectangles dessinés à différents endroits de l'écran. Chaque fois qu'une touche est tapée, les couleurs INK sont modifiées de telle sorte que le rectangle affiché précédemment passe dans la couleur du fond (il est donc effacé), tandis que l'autre prend la couleur du dessin :

```
10 MODE 1

20 x=100: y=100

30 xd=50: yd=100

39 REM PEN 1 pour la couleur du fond

40 INK 1,1

49 REM dessin d' un rectangle avec PEN 1

50 colour=1: GOSUB 1000

60 x=300: y=300

70 xd=100: yd=50

79 REM PEN 2 pour la couleur du fond

80 INK 2,1

89 REM dessin du second rectangle avec P
EN 2

90 colour= 2: GOSUB 1000

100 continue=1

109 REM affichage jusqu' a ce que 'e' soi
t tape

110 WHILE reponse$<>"e"

119 REM PEN 1 couleur principale PEN 2 c
ouleur du fond

120 INK 1,24
```

```
130 INK 2,1
140 reponse$=""
150 WHILE reponse$=""
160 reponse$=LOWER$(INKEY$)
170 WEND
179 REM PEN 2 couleur principale PEN 1 c
ouleur du fond
180 INK 1,1
190 INK 2,24
200 reponse$=""
209 REM attente de la frappe d'une touch
e
210 WHILE reponse$=""
220 reponse$=LOWER$(INKEY$)
230 WEND
240 WEND
249 REM retour a la couleur initiale
250 INK 1,24
260 INK 2,20
270 END
356 RUN"
1000 MOVE x,y
1010 DRAWR xd,0,colour
1020 DRAWR 0,yd
1030 DRAWR -xd,0
1040 DRAWR 0,-yd
1050 RETURN
```

Cet affichage est très rapide parce que les figures dessinées auparavant sont affichées instantanément. Ce principe pourrait être étendu de façon à animer une séquence de dessins en les élaborant dans la couleur du fond, puis en les affichant successivement. Cependant, cette méthode ne peut pas être utilisée telle quelle pour l'animation du dessin de chien, parce que le chevauchement des images pose un problème :

```
59 REM chevauchement du second rectangle
60 x=120 : y=100
```

Une partie du rectangle manque quand les figures se chevauchent ; une ligne est commune aux deux figures et l'attribution de la couleur du fond pour un rectangle supprime une partie de l'autre.

Si les lignes ne se chevauchent pas, la modification des couleurs (INK) génère une animation rapide et réussie, surtout en mode 0 dans lequel on dispose de seize couleurs différentes. Il est possible de dessiner jusqu'à quinze images dans la couleur du fond, de les afficher successivement en leur attribuant la couleur d'encre adéquate puis de leur attribuer à nouveau la couleur du fond :

```
10 MODE 0
19 REM position de depart
20 startx=260: starty=180
29 REM taille des cotes
30 lengthx=20: lengthy=20
39 REM difference de taille entre les re
ctangles
40 incx=8: incy=6
50 startink=1: endink=15
60 GOSUB 1000
70 GOSUB 2000
80 END
```

```
998 REM dessin de 15 rectangles dans la
couleur du fond

999 REM imbriques les uns dans les autre
s

1000 FOR count=startink TO endink

1010 INK count,1

1020 movex=count*incx

1030 movey=count*incy

1040 sidex=lengthx+2*movex

1050 sidey=lengthy+2*movey

1060 MOVE startx-movex,starty-movey

1070 DRAWR sidex,0,count

1080 DRAWR 0,sidey

1090 DRAWR -sidex,0

1100 DRAWR 0,-sidey

1110 NEXT

1120 RETURN

1999 REM

2000 continue=1

2010 startink=1:nextink=2

2019 REM on continue

2020 WHILE continue=1

2029 REM attente de la frappe d'une touc
he

2030 reponse$=""

2040 WHILE reponse$=""

2050 reponse$=INKEY$

2060 WEND
```

```
2069 REM le rectangle precedent prend la
 couleur du fond

2070 INK startink,1

2079 REM le rectangle suivant prend la c
ouleur principale

2080 INK nextink,24

2088 REM incrementation des INK afin qu'
au prochain cycle

2089 REM les INK actuelles prennent la c
ouleur du fond et les nouvelles la coule
ur principale

2090 startink=(startink+1) MOD 16

2100 IF startink=0 THEN startink=1

2110 nextink=(nextink+1) MOD 16

2120 IF nextink=0 THEN nextink=1

2130 WEND

2140 RETURN
```

Cette méthode est particulièrement efficace si l'affichage est cyclique comme dans cet exemple ; elle a cependant certaines limites car on ne peut pas toujours éviter le chevauchement de lignes.

L'option EXOR évite que le dessin d'une ligne perturbe les lignes déjà présentes :

```
 15 PRINT CHR$(23) CHR$(1);
265 PRINT CHR$(23) CHR$(0);
```

C'est un succès partiel ; les deux rectangles sont complètement visibles, mais les points qui leur sont communs apparaissent en rouge. Cela est engendré par la manière dont les couleurs sont représentées en mode 1.

Seules quatre couleurs peuvent être affichées simultanément en mode 1, parce que la couleur de chaque point est déterminée par un code sur deux bits. Étant donné qu'il existe seulement quatre combinaisons différentes avec deux bits, seules quatre couleurs sont pos-

sibles. Ces codes ne changent jamais bien que l'on puisse employer la commande INK pour que l'ordinateur interprète n'importe quel code comme une couleur différente. Par exemple, si on utilisait la commande INK 1,6, l'ordinateur afficherait un pixel rouge et non jaune.

00000000	Bleu	Effectivement, la couleur
00000001	Jaune	de n'importe quel point
00000010	Cyan	est codée sur deux bits.
00000011	Rouge	

Figure 5.2 : Code binaire des quatre couleurs en mode 1.

Dans cet exemple, on dessine deux rectangles, l'un à l'aide de la commande INK 1 (code 01) et l'autre à l'aide de la commande INK 2 (code 10). L'option EXOR associe l'ancienne et la nouvelle combinaison des bits selon une règle simple : si les bits sont identiques, le résultat est 0 ; si les bits sont différents, le résultat est 1. Quand les deux rectangles se chevauchent, on obtient le résultat de la Figure 5.3. Le code 11 indique PEN 3, établi à rouge en mode1, et la ligne est affichée en rouge. Cela soulève un problème. Si l'on utilise la commande INK pour réinitialiser PEN 3 de telle sorte qu'il génère la couleur jaune plutôt que rouge, la zone de chevauchement devient invisible :

```
 11 REM utiliser 3 pour l'encre jaune
 12 INK 3,24
261 REM 3 rétablit l'encre jaune à la fin
262 INK 3,6
```

Bien que les rectangles se chevauchent, ils peuvent être affichés alternativement en commutant les paramètres de la commande INK. C'est la base d'une méthode générant une animation plus dynamique :

1. La première figure est dessinée dans la couleur du dessin et affichée.
2. La seconde figure est dessinée à l'aide de la commande PEN établie dans la couleur du fond.
3. Les couleurs INK sont commutées pour que la seconde figure soit affichée et la première occultée.
4. La première figure est effacée de son ancienne position.

5. Une nouvelle figure est dessinée dans la couleur du fond.
6. Les couleurs sont commutées pour que la nouvelle figure soit affichée et la seconde occultée...

...et ainsi de suite jusqu'à ce que l'animation soit complète. La seule difficulté vient du chevauchement des figures ; il nous faut donc tracer ou effacer une figure sans en affecter une autre. Nous allons examiner ce problème de plus près.

On suppose que l'on travaille en mode 1 et que les deux figures successives sont dessinées respectivement à l'aide des commandes PEN 1 et PEN 2. Cela nous donne une idée des quatre codes couleur exprimés sur deux bits. Il n'est pas nécessaire de s'occuper des couleurs réellement affichées sur l'écran, puisque la commande INK permet de choisir la couleur générée par n'importe quelle commande PEN. Nous nous intéresserons plus particulièrement aux codes deux bits et à leur fonctionnement.

	01	Un point d'une ligne jaune
EXOR	10	chevauché par une ligne cyan
	11	donne un point rouge

Figure 5.3

Pour effacer une ligne quelconque dessinée à l'aide de la commande PEN 1, il faut convertir le code 01 en 00, couleur du fond. Cela peut être réalisé à l'aide de la commande EXOR. Cependant, certains points de la ligne peuvent chevaucher des points de la ligne de la seconde figure, le code est donc 11. N'importe quel chevauchement produit une couleur correspondant à PEN 2. L'instruction EXOR ne résout pas tous les problèmes ; si un point se trouve à l'intersection de deux lignes, l'effacement d'une ligne fait disparaître ce point, mais l'effacement de la seconde ligne le fait réapparaître. On peut obtenir le résultat désiré d'une façon légèrement différente en utilisant un autre mode de dessin de ligne disponible sur l'Amstrad et que nous allons aborder maintenant.

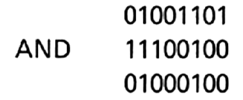
La couleur peut être déterminée à l'aide d'une instruction AND (ET logique) qui associe l'ancien et le nouveau code du point. Cela est réalisé grâce aux codes de contrôle PRINT CHR$(23) CHR$(2), qui entraîne l'utilisation de l'instruction AND pour toutes les commandes graphiques ultérieures.

	01	Un point sur une ligne jaune
EXOR	01	croisé par une ligne jaune
	00	donne un point bleu, couleur du fond.

Figure 5.4

	11	Un point rouge
EXOR	01	croisé par une ligne jaune
	10	donne un point cyan.

Figure 5.5



	01001101
AND	11100100
	01000100

Figure 5.6 : Exemple de résultat généré par la commande AND.

Après une commande AND, le bit est établi à 1 seulement si les bits de rang correspondants sont égaux à 1 ; sinon, le code est établi à 0. On peut effacer les points dessinés avec la commande PEN 1 en leur associant le code 10 à l'aide d'une instruction AND (voir la Figure 5.7). Dans ce cas, les points de chevauchement restent dans la couleur correcte comme le montre la Figure 5.8. De même, on peut effacer les points dessinés avec la commande PEN 2 en leur associant le code 01 à l'aide de l'instruction AND (voir la Figure 5.9).

	01	Un point sur une ligne jaune
AND	10	croisé par une ligne cyan
	00	donne un point bleu, couleur du fond.

Figure 5.7

	11	Un point rouge
AND	10	croisé par une ligne cyan
	10	donne un point cyan.

Figure 5.8

	10	Un point cyan
AND	01	croisé par une ligne jaune
	00	donne un point bleu, couleur du fond.

Figure 5.9

Nous allons maintenant reprendre le dessin sans affecter les points déjà tracés. On suppose que le crayon utilisé est PEN 1. Les résultats désirés pour les différents points apparaissent Figure 5.10.

Code couleur du point	Code désiré après croisement du point et de la ligne dessinée avec PEN 1
00000000	00000001
00000001	00000001
00000010	00000011
00000011	00000011

Figure 5.10

Aucune des deux commandes EXOR et AND ne donne le résultat recherché. Il existe une autre instruction logique que nous ne connaissons pas encore, l'instruction OR. Elle est établie par les codes de contrôle PRINT CHR$(23) CHR$(3) et a pour résultat 1 si un au moins des bits a pour valeur 1. On peut associer au code couleur le code 01 à l'aide de l'instruction OR ou lui associer le code 10 comme dans la Figure 5.12.

	01001101
OR	11100100
	11101101

Figure 5.11 : Exemple de résultat d'une instruction OR.

	00		01		10		11
OR	10	OR	10	OR	10	OR	10
	10		11		10		11

Figure 5.12

Il est maintenant possible d'identifier une séquence de tracés et de couleurs qui dessine ou efface à la demande, comme le montre le programme suivant.

```
10 MODE 1
20 INK 3,24
30 DEFINT c,s,t,x,y
40 x=100: y=100
50 x1=100: y1=200
60 colour=1: colour1=24
70 type=3: shade=1
80 GOSUB 1000
90 GOSUB 2000
100 type=3: shade=2
110 WHILE x<639
120 x=x+4: x1=x1+4
130 GOSUB 2000
140 GOSUB 1000
150 x=x-4: x1=x1-4
160 GOSUB 2000: x=x+4: x1=x1+4
170 IF shade=2 THEN shade=1 ELSE shade=2
180 WEND
189 REM retour des INK au mode initial
190 INK 1,24
200 INK 2,20
210 INK 3,6
220 END
```

```
998 REM la couleur d'avant plan devient
couleur du fond

999 REM celle du fond la couleur d'avant
 plan

1000 IF colour=1 THEN colour=24:colour1=
1 ELSE colour=1:colour1=24

1010 INK 1,colour

1020 INK 2,colour1

1030 RETURN

1998 REM routine effacement/ecriture

1999 REM

2000 PRINT CHR$(23);CHR$(type);

2010 MOVE x,y

2020 DRAW x1,y1,shade

2030 DRAW x1+50,y1

2040 DRAW x+50,y1

2050 DRAW x,y

2060 IF type=2 THEN type=3 ELSE type=2

2070 RETURN
```

La ligne 100 replace temporairement les coordonnées du triangle dans leur position initiale pour qu'il puisse être effacé pendant qu'il se trouve dans la couleur du fond. La ligne 120 commute les couleurs qui servent au dessin ou à l'effacement et la ligne 2030 commute les modes de tracé OR et AND.

On peut appliquer cette technique à l'animation du dessin du chien.

```
10 MODE 1

14 REM on utilise des entiers pour plus
de vitesse

15 DEFINT c,s,t,x,y
```

```
20 GOSUB 1000

30 GOSUB 2000

40 PRINT CHR$(23)CHR$(0);

50 END

998 ON dessine dans deux plans de 26 coo
rdonnees

999 REM pour avoir deux images du chien

1000 DIM x(100),y(100)

1010 FOR count=1 TO 52

1020 READ x(count),y(count)

1030 NEXT

1040 RETURN

1050 DATA 0,0,20,40,20,40,10,80,10,80,0,
120,20,10,35,50,35,50,10,80,10,80,70,80

1060 DATA 60,0,80,40,80,40,70,80,80,10,9
5,50,95,50,70,80,70,80,90,140,90,140,110
,120,110,120,80,110

1070 DATA 5,10,25,40,25,40,15,80,15,80,1
0,120,0,15,30,50,30,50,15,80,15,80,75,80

1080 DATA 75,80,85,40,85,40,65,10,75,80,
90,50,90,50,60,15,75,80,100,130,100,130,
120,110,120,110,90,100

1999 REM on utilise INK 3 pour corriger
la couleur pour le ou exclusif

2000 INK 3,24

2010 colour=1:colour1=24

2020 type=3:shade=1

2029 REM on dessine la figure a la posit
ion initiale

2030 GOSUB 4000

2040 GOSUB 5000
```

```
2050 type=3: shade=2

2060 xinc=0

2070 WHILE x(1)+xinc<639

2079 REM mise a jour de xinc pour positi
onner la nouvelle figure

2080 xinc=xinc+20

2089 REM on dessine la nouvelle figure d
ans la couleur du fond

2090 GOSUB 5000

2100 GOSUB 4000

2109 REM on efface la figure precedente
en utilisant la couleur de fond

2110 xinc=xinc-20

2120 GOSUB 5000

2130 xinc=xinc+20

2140 IF shade=2 THEN shade=1 ELSE shade=
2

2150 WEND

2160 RETURN

2999 ON dessine le chien comme une serie
 de points continues

3000 FOR count=start TO start+25 STEP 2

3010 MOVE x(count)+xinc, y(count)

3020 DRAW x(count+1)+xinc, y(count+1), sha
de

3030 NEXT

3040 RETURN

3999 ON inverse les couleurs

4000 IF colour=1 THEN colour=24: colour1=
1 ELSE colour=1: colour1=24
```

```
4010 INK 1,colour

4020 INK 2,colour1

4030 RETURN

4999 REM on change de mode suivant que l
'on efface ou que l'on dessine

5000 PRINT CHR$(23)CHR$(type);

5010 IF type+shade=4 THEN start=1 ELSE s
tart=27

5020 GOSUB 3000

5030 IF type=2 THEN type=3 ELSE type=2

5040 RETURN
```

EXERCICES

1. Dessiner une série de cercles dans des positions et des couleurs différentes, puis simuler des ballons qui rebondissent en affichant alternativement chaque cercle dans les couleurs du fond et du dessin.
2. Créer deux images de petits bonshommes stylisés qui ont alternativement les bras le long du corps et levés à l'horizontale.

CRÉATION DES COULEURS DU DESSIN ET DU FOND

Nous allons étudier maintenant une autre conséquence de la manipulation des couleurs INK. Ce programme dessine deux rectangles qui se chevauchent ; cette fois-ci, les rectangles sont entièrement colorés, l'un en jaune et l'autre en bleu :

```
10 MODE 1

19 REM jaune pour la couleur d'avant pla
n

20 INK 3,24

30 PRINT CHR$(23)CHR$(1);
```

```
40 x=100: y=100
50 xd=50: yd=100
60 INK 1,24
70 colour=1: GOSUB 1000
80 x=120: y=100
90 xd=100: yd=50
100 INK 2,20
110 colour=2: GOSUB 1000
120 PRINT CHR$(23)CHR$(0);
130 END
999 REM on rempli le rectangle
1000 FOR xcoord=x TO x+xd STEP 2
1010 MOVE xcoord, y
1020 DRAWR 0, yd, colour
1030 NEXT
1040 RETURN
```

Après déroulement du programme, il n'est pas étonnant de constater que l'un des rectangles masque une partie du second. Ce qui est surprenant, c'est que le rectangle jaune dessiné en premier, cache une partie du rectangle bleu dessiné en second. Cela est la conséquence de la ligne 20 qui contient les indications suivantes : ''*Faire apparaître toute zone de chevauchement en jaune*''. Nous avons décidé que le jaune serait la couleur du dessin et le bleu la couleur du fond, et que, dans le cas d'un chevauchement, le jaune serait prioritaire.

Si on modifie la ligne 20 pour que toute zone de chevauchement soit affichée en bleu, le rectangle bleu cache partiellement le rectangle jaune :

```
19 REM coloration du dessin en bleu
20 INK 3,20
```

La possibilité de donner une priorité aux couleurs est très utile dans les jeux. On peut, en établissant les couleurs INK de manière appropriée, organiser les couleurs de telle sorte qu'une figure puisse être superposée à une autre sans l'effacer. De même, une figure peut passer "derrière" des zones tracées dans la couleur du dessin et ressortir de l'autre côté parfaitement intacte.

Dans le Chapitre 2, nous avons étudié une des méthodes d'animation les plus simples en utilisant l'instruction TAG et en affichant le caractère sur sa nouvelle position :

```
10 MODE 1
20 x=230: y=130
29 REM dessin et coloriage d' un rectangl
e
30 FOR xcoord=x TO x+100 STEP 2
40 MOVE xcoord, y
50 DRAWR 0,100,1
60 NEXT
70 xprint=0: yprint=180
79 REM curseur texte ou graphique
80 TAG
89 REM on affiche le caractere a des pos
itions succesives
90 FOR xcoord=xprint TO 400 STEP 2
100 MOVE xcoord, yprint
110 PRINT CHR$( 233);
120 NEXT
130 END
```

Le caractère efface une partie du fond pendant son déplacement. L'utilisation d'un caractère "non limité" à gauche donne des résultats encore plus mauvais :

```
109 REM caractere "non limite" à gauche
; une fleche
110 PRINT CHR$(243)
```

La solution consiste à utiliser l'instruction TAG en combinaison avec l'instruction EXOR. Cependant, le simple affichage du caractère ne suffit pas :

```
10 MODE 1

20 x=230: y=130

29 REM dessin et coloriage d'un rectangl
e

30 FOR xcoord=x TO x+100 STEP 2

40 MOVE xcoord, y

50 DRAWR 0, 100, 1

60 NEXT

70 xprint=0: yprint=180

71 REM

75 PRINT CHR$( 23) CHR$( 1);

80 TAG

89 REM affichage d'un caractere a des po
sitions successives

90 FOR xcoord=xprint TO 400 STEP 2

100 MOVE xcoord, yprint

110 PRINT CHR$( 243);

120 NEXT

130 TAGOFF
```

```
140 PRINT CHR$(23)CHR$(0);

150 END
```

Le résultat est encore pire parce que le caractère est associé, par la commande EXOR, à son image déplacée d'un pixel vers la droite ; le résultat de cette combinaison ne donne pas le caractère initial.

Nous allons concevoir notre propre caractère flèche avec une limite à gauche, comme dans la Figure 5.13. La flèche sera affichée à partir de la position de départ sur l'écran ; ensuite, les caractères seront affichés (en combinaison EXOR) successivement sur chaque pixel placé à droite du pixel en cours. Il faut d'abord définir un second caractère qui, combiné avec le caractère initial, donne une copie du caractère initial déplacée d'un pixel vers la droite. Cela est plus facile à comprendre sur un diagramme (voir la Figure 5.14).

On peut trouver la définition du caractère flèche en examinant la définition de la flèche initiale, rangée par rangée, et en calculant quelles valeurs il faut lui associer pour la reproduire. Cela est plus facile à réaliser en binaire, comme le montre la Figure 5.15, dans laquelle nous calculons le premier nombre nécessaire à la définition du caractère. La première ligne du caractère modifié par EXOR doit être 24. On pourrait continuer de la même manière, mais il existe un moyen plus rapide ; la combinaison de deux nombres binaires par une instruction EXOR donne un résultat particulier. La combinaison du nombre initial avec le résultat obtenu donne le second nombre. Par conséquent, la nouvelle définition peut être trouvée plus facilement en combinant chaque ligne du caractère initial avec la même ligne déplacée d'un pixel vers la droite (voir la Figure 5.17). Il est possible d'incorporer les deux caractères dans un programme pour démontrer que la méthode de combinaison laisse le fond intact :

```
84 REM on definie la premiere fleche a a
fficher

85 SYMBOL 240,8,12,14,127,127,14,12,8
```

```
86 REM maintenant on definie la fleche e
n faisant un ou exclusif

87 SYMBOL 241,24,20,18,129,129,18,20,24

90 FOR xcoord=xprint TO 400 STEP 2

100 MOVE xcoord,yprint

108 REM affichage de la fleche originale
 a la premiere position

109 REM puis la seconde pour la placer a
 une nouvelle position

110 IF xcoord=xprint THEN PRINT CHR$(240
);ELSE PRINT CHR$(241);

120 NEXT

130 TAGOFF

140 PRINT CHR$(23)CHR$(0);

150 END
```

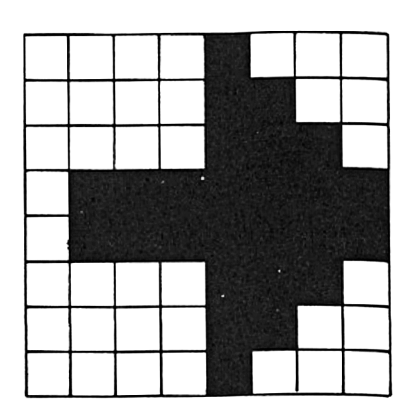


Figure 5.13 : Caractères avec une limite.

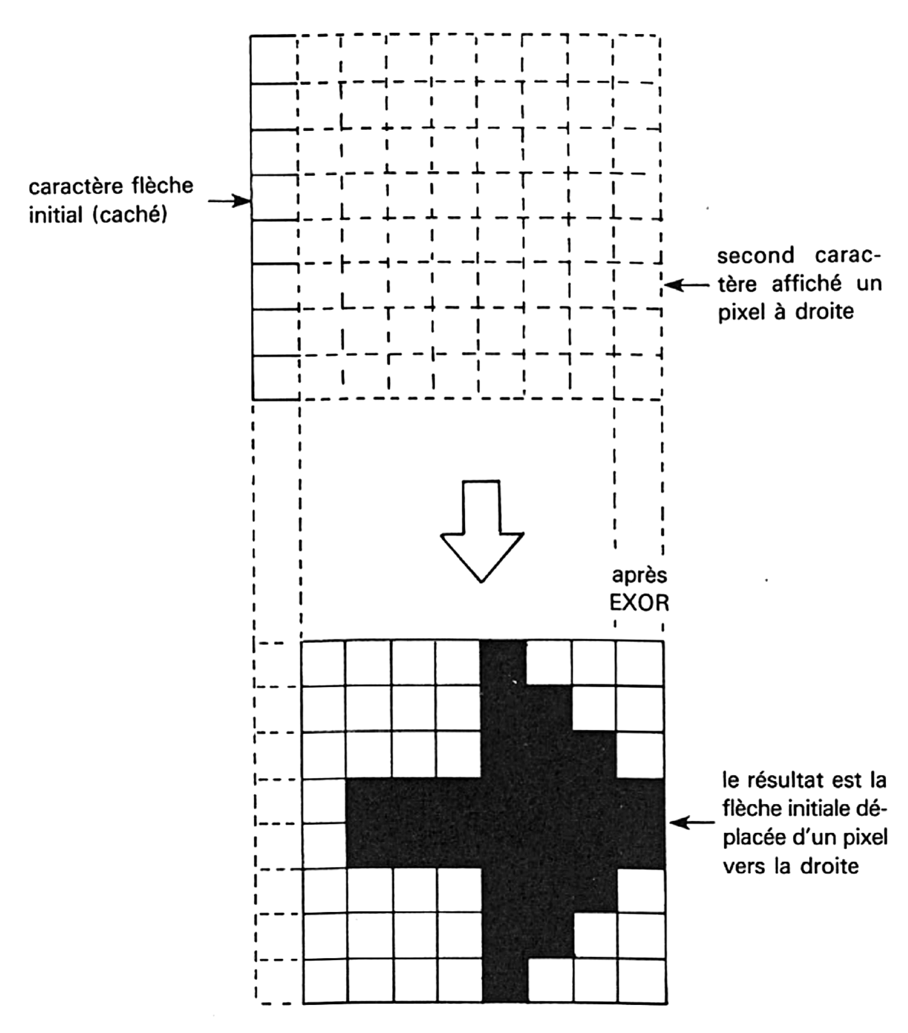


Figure 5.14 : Copie, à l'aide de la commande EXOR d'un premier caractère affiché un pixel vers la droite.

	00001000	Nombre binaire correspondant à la première rangée du caractère flèche (8)
EXOR	00011000	associé à ce nombre binaire (24)
	00001000	pour donner le premier code déplacé vers la droite (8).

Figure 5.15

	00001000	Nombre binaire correspondant à la première rangée du caractère flèche (8)
EXOR	00001000	Résultat que l'on veut obtenir après combinaison entre 8
	00011000	et 24.

Figure 5.16

	00001100	Nombre binaire correspondant à la première rangée du caractère flèche (12)
EXOR	00001100	Résultat que l'on veut obtenir après combinaison entre 12
	00010100	et 20.

	00001110	Nombre binaire correspondant à la troisième rangée du caractère flèche (14)
EXOR	00001110	Résultat que l'on veut obtenir après combinaison entre 12
	00010010	et 18.

...et ainsi de suite pour les autres rangées de la définition de caractères.

Figure 5.17 : Pour trouver la définition de caractères, il faut générer une copie du caractère flèche déplacée d'un pixel sur la droite.

Cela donne la possibilité de générer toutes sortes d'effets dans des programmes de jeux. On peut créer, en combinant les instructions TAG et EXOR et en modifiant les couleurs INK, une gamme de caractères de jeux qui se comportent de différentes manières. Par exemple, les protagonistes du jeu peuvent se déplacer parmi les blocs jaunes, mais pas parmi les bleus ; les fantômes qui les poursuivent peuvent seulement traverser les bleus et rien n'arrête le "super fantôme".

Lors de la création d'un caractère, il ne faut pas oublier d'ajouter une bordure d'un pixel, sous peine de voir une traînée laissée par son déplacement. Cette bordure doit être placée sur le côté opposé à la direction du déplacement. Dans le cas de la flèche, la bordure était placée à gauche, car celle-ci se déplaçait vers la droite.

Si on décide de déplacer une figure dans plusieurs directions, il faut définir un caractère qui sera combiné par l'intermédiaire d'une instruction EXOR pour chaque direction de déplacement et l'utiliser pour l'affichage.

Il n'est pas obligatoire de se confiner à des caractères uniques, bien qu'ils soient plus faciles à manipuler. Ce programme dessine une "voiture" composée de trois caractères et de leurs trois caractères EXOR correspondants. La voiture se déplace sur l'écran derrière les blocs jaunes et devant les blocs cyan :

```
10 MODE 1

20 GOSUB 1000

30 GOSUB 2000

40 GOSUB 3000

50 TAGOFF

60 PRINT CHR$(23)CHR$(0);

70 END

998 REM la couleur d'avant plan est jaun
e

1000 INK 3,24

1010 y=100:colour=1

1020 FOR x=100 TO 500 STEP 50

1030 GOSUB 4000

1040 NEXT

1050 RETURN

2000 SYMBOL 240,0,15,28,124,127,18,12,0

2010 SYMBOL 241,0,255,120,120,255,255,0,
0

2020 SYMBOL 242,0,128,192,254,254,72,48,
0

2030 SYMBOL 243,0,16,36,132,128,55,20,0
```

```
2040 SYMBOL 244,0,0,137,137,0,1,0,0

2050 SYMBOL 245,0,128,64,2,2,216,80,0

2060 car$=CHR$(240)+CHR$(241)+CHR$(242)

2070 exorcar$=CHR$(243)+CHR$(244)+CHR$(2
45)

2080 RETURN

3000 PRINT CHR$(23)CHR$(1);

3010 TAG

3020 xprint=0:yprint=116

3030 FOR xcoord=xprint TO 550 STEP 2

3040 MOVE xcoord,yprint

3050 IF xcoord=xprint THEN PRINT car$;EL
SE PRINT exorcar$;

3055 r$="":WHILE r$="":r$=INKEY$:WEND

3060 NEXT

3070 RETURN

4000 IF colour=1 THEN colour=2 ELSE colo
ur=1

4010 FOR xcoord=x TO x+30 STEP 2

4020 MOVE xcoord,y

4030 DRAWR 0,100,colour

4040 NEXT

4050 RETURN
```

La ligne 1000 est nécessaire pour attribuer la couleur jaune au chevauchement de la voiture cyan et du bloc jaune. Si l'on remplace la commande INK 3,24 par la commande INK 3,6 (couleur normale de PEN 3), la voiture devient rouge quand elle traverse le bloc jaune.

La voiture reprend la couleur du fond quand elle passe sur les blocs cyan, parce que les deux couleurs combinées sont cyan et que la

combinaison d'un nombre binaire avec lui-même donne toujours la valeur 0, couleur du fond en cours.

	0010	Un point cyan
EXOR	0010	recouvert par un autre point cyan
	0000	donne un point bleu, couleur du fond.

Figure 5.18

On pourrait résoudre ce problème en traçant des blocs cyan dans une couleur différente et en établissant la couleur INK de telle sorte que le chevauchement de la voiture et du bloc ne génère pas cette modification de couleur. Mais en mode 1, nous ne disposons que de quatre couleurs qui seront rapidement employées. Il existe beaucoup plus de possibilités en mode 0, car on dispose alors de seize couleurs qui permettent de redéfinir les couleurs PEN pour obtenir l'effet désiré.

```
10 MODE 0

998 REM INK 3 couleur jaune cyan/jaune d
onnent jaune

999 REM jaune est la couleur d'avant pla
n

1000 INK 3,24

1001 REM INK 6 couleur cyan cyan/blanc d
onnent cyan

1002 REM blanc est la couleur de fond

1003 INK 6,20

2071 MOVE 0,0

2072 DRAW 0,0,2

4000 IF colour=1 THEN colour=4 ELSE colo
ur=1
```

En mode 0, chaque couleur est représentée par quatre bits. La voiture est affichée à l'aide des instructions PEN 2 et EXOR ; elle peut chevaucher soit un bloc jaune (dessiné par PEN 1), soit un bloc rouge

(dessiné par PEN 4). Nous devons donc réinitialiser PEN 3, lignes 1000 et 1001, pour établir la couleur jaune (de telle sorte que la voiture semble passer derrière les blocs jaunes), et PEN 6 pour établir la couleur cyan (pour que la voiture passe devant les blocs rouges). Il est possible d'inverser la priorité des couleurs pour que la voiture passe devant les blocs jaunes et derrière les blocs rouges :

```
998 REM INK 3 cyan cyan/jaune donnent cy
an

999 REM jaune est la couleur de fond

1000 INK 3,20

1001 REM INK 6 blanc cyan/blanc donnent
blanc

1002 REM blanc est la couleur d'avant pl
an

1003 INK 6,26
```

	0010	La voiture cyan
EXOR	0001	qui chevauche un bloc jaune
	0011	génère le code rouge pour PEN 3.
	0010	La voiture cyan
EXOR	0100	qui chevauche un bloc blanc
	0110	génère un code bleu pour PEN 6.

Figure 5.19

L'emploi de la commande TAG pose quelques problèmes ; l'affichage est exécuté dans la couleur du premier plan. Il n'est plus possible de modifier cette couleur à l'aide d'une commande PEN, car après une instruction TAG, l'ordinateur ne modifie les couleurs qu'en obéissant à une commande graphique.

Avant que la voiture soit affichée dans la couleur cyan, la couleur graphique doit être commutée de blanc à cyan. C'est la raison d'être des lignes 2071 et 2072, sans lesquelles la voiture serait affichée dans la couleur en cours, c'est-à-dire blanc.

0000	Bleu	Couleur du 4e plan.
0001	Jaune	Couleur du 3e plan.
0010	Cyan	Couleur du 2e plan.
0011	Cyan	Couleur du 2e plan cachant le 3e plan.
0100	Blanc	Couleur du 1er plan.
0101	Blanc	Couleur du 1er plan cachant le 2e plan.
0111	Blanc	Couleur du 1er plan cachant le 2e plan cachant les autres plans.

Figure 5.20 : Création des couleurs des différents plans en établissant les paramètres appropriés pour la commande INK.

Le mode 0 offre de nombreuses possibilités grâce à la gamme de couleurs dont il dispose. Par exemple, il est possible d'établir les couleurs des premier, deuxième et troisième plans, comme le montre la Figure 5.20. Dans cette figure, le quatrième bit ayant pour valeur 1 indique la couleur jaune, couleur du troisième plan ; le troisième bit établi à 1 indique la couleur cyan, couleur du deuxième plan ; le deuxième bit ayant pour valeur 1 indique la couleur blanche, couleur du premier plan. D'autres combinaisons indiquent une couleur qui en occulte une autre.

L'utilisation de la commande EXOR permet de tracer et d'effacer des lignes dans n'importe quelle couleur sans perturber le chevauchement ou les lignes cachées dans d'autres couleurs. Nous allons étudier par exemple l'effet d'une commande EXOR sur une ligne blanche du premier plan cachant une ligne cyan du deuxième plan, qui cache elle-même une ligne jaune du troisième plan. On commence par effacer la ligne blanche (voir la Figure 5.21) ; que se passerait-il si nous avions effacé la ligne cyan (voir la Figure 5.22) ?

	0111	Un point blanc cachant un point cyan
EXOR	0100	cachant un point jaune.
	0011	L'effacement du premier plan blanc révèle le deuxième plan cyan.

Figure 5.21

	0111	Un point blanc cachant un point cyan
EXOR	0010	cachant un point jaune.
	0101	Après effacement du point cyan du second plan, le premier plan blanc ne cache plus que le troisième plan jaune.

Figure 5.22

En examinant les résultats de l'effacement d'une ligne, il est possible de déterminer quelles sont les commandes INK nécessaires à l'effacement d'une ligne quelconque sans perturbation apparente pour les autres. Cela est visible dans le programme suivant, qui trace et colore un rectangle blanc en premier plan, sur un rectangle cyan en deuxième plan, qui se trouve lui-même sur un rectangle jaune en troisième plan :

```
10 MODE 0
20 GOSUB 1000
30 GOSUB 2000
40 GOSUB 3000
50 PRINT CHR$(23)CHR$(0);
60 MODE 1
70 END
999 REM initialisation des crayons (PEN)
1000 INK 3,20
1010 INK 5,26
1020 INK 6,26
1030 INK 7,26
1040 RETURN
1999 REM on dessine 3 rectangles
2000 xyellow=100: yyellow=40: sideyellow=3
00
```

```
2010 xcyan=150: ycyan=80: sidecyan=220

2020 xwhite=200: ywhite=120: sidewhite=140

2029 REM

2030 PRINT CHR$(23)CHR$(1);

2040 colour=1: x=xyellow: y=yyellow: side=s
ideyellow

2050 GOSUB 4000

2060 colour=2: x=xcyan: y=ycyan: side=sidec
yan

2070 GOSUB 4000

2080 colour=4: x=xwhite: y=ywhite: side=sid
ewhite

2090 GOSUB 4000

2100 RETURN

3000 WINDOW 1,20,25,25

3009 REM entree des commandes

3010 WHILE reponse$<>"e"

3020 INPUT"Commande (y/c/w/e)",reponse$

3029 REM dessin ou effacement du rectang
le jaune

3030 IF reponse$="y" THEN colour=1: x=xye
llow: y=yyellow: side=sideyellow: GOSUB 400
0

3039 REM dessin ou effacement du rectang
le cyan

3040 IF reponse$="c" THEN colour=2: x=xcy
an: y=ycyan: side=sidecyan: GOSUB 4000

3049 REM dessin ou effacement du rectang
le blanc

3050 IF reponse$="w" THEN colour=4: x=xwh
ite: y=ywhite: side=sidewhite: GOSUB 4000
```

```
3890 WEND
3900 RETURN
4000 FOR xcoord=x TO x+side STEP 4
4010 MOVE xcoord, y
4020 DRAWR 0, side, colour
4030 NEXT
4040 RETURN
```

La ligne 3020 permet de dessiner ou d'effacer les rectangles jaune, cyan et blanc en entrant respectivement les lettres j, c et b (f pour fin). Ce programme peut surprendre : en effet, on peut effacer ou dessiner le rectangle jaune sans toucher aux deux rectangles qui sont ''devant'' lui.

EXERCICES

1. Concevoir un ''bateau'' à l'aide d'un ou de plusieurs caractères, ainsi qu'un caractère EXOR approprié pour qu'il puisse être déplacé sur l'écran.
2. Améliorer le programme précédent pour faire naviguer le bateau dans de l'eau bleue et le faire passer devant un certain nombre de récifs qui sortent de l'eau.
3. Ajouter des îles derrière lesquelles passe le bateau.
4. Faire naviguer un autre bateau dans la direction opposée. Quand les bateaux se croisent, l'un d'entre eux passe derrière l'autre.
5. Modifier l'ordre de priorité des couleurs dans le programme de conception de rectangles pour que la couleur jaune devienne la couleur du premier plan et la couleur blanche celle du fond. Seul le rectangle jaune est visible au départ et le rectangle blanc n'apparaît qu'après effacement des deux autres.
6. Ajouter un quatrième rectangle dans une couleur qui devient la couleur du premier plan. Ajuster les couleurs définies par l'instruction INK de telle sorte que n'importe quel rectangle puisse être dessiné ou effacé sans affecter les autres.

6

... ET EFFETS ARTISTIQUES

Dans le Chapitre 4, nous avons développé un programme qui permet à l'utilisateur de dessiner sur l'écran et de sauvegarder le dessin obtenu dans un fichier pour un emploi ultérieur. Nous allons l'améliorer dans ce chapitre et étendre ses possibilités pour pouvoir dessiner et colorer des formes standard.

SÉLECTION D'OPTIONS A PARTIR DU MENU

Le programme décrit ci-dessus reposait entièrement sur les entrées à partir du clavier. La plupart des programmes de ce type affichent un *menu* sur l'écran. Un menu présente à l'utilisateur une gamme d'options disponibles ; les options classiques sont par exemple la modification de l'échelle, la sélection d'une nouvelle couleur ou le dessin d'un cercle ou d'un rectangle (voir la Figure 6.1).

Les sélections dans le menu sont réalisées par le déplacement du curseur à la position appropriée sur l'écran. L'ordinateur note la position du curseur et met en œuvre le choix correspondant. Ce processus rend le programme beaucoup plus facile à utiliser si les indications du menu sont explicites. Il n'est plus nécessaire de savoir quelle touche du clavier modifie la couleur de la ligne ou permet de tracer un cercle.

Cependant, si on ne dispose pas d'un autre périphérique d'entrée tel qu'une manette de jeu, certains contrôles doivent quand même être effectués à partir du clavier ; par exemple, le déplacement du curseur à un endroit précis de l'écran.

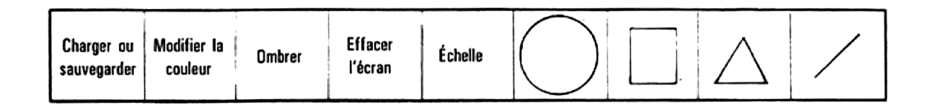


Figure 6.1 : Menu classique d'un programme de dessin.

Le nouveau programme comporte des options de sélection de la couleur de la ligne (huit couleurs y compris la couleur du fond). Il offre également la possibilité de tracer des lignes uniques, un cercle, un triangle ou un rectangle. Une autre option proposera de colorer ou non une figure.

La sélection d'un choix dans le menu est indiquée par la frappe de la touche f (point fixé) dès que le curseur passe dans la zone de l'option choisie. L'ordinateur émet ensuite un bip. Il est parfois difficile de se souvenir du mode de dessin ou de la couleur en cours. C'est la raison pour laquelle il existe un système de rappel. Celui-ci consiste en un caractère affiché dans l'angle inférieur gauche de l'écran, qui indique la situation en cours. Le menu apparaît également en bas de l'écran.

```
10 MODE 0

20 GOSUB 1000

30 GOSUB 2000

40 END

1000 startx=320: starty=200

1010 x=startx: y=starty

1020 foregroundcolour=1

1030 linedraw=0

1037 REM affichage d' un symbole en bas a
 gauche

1038 REM afin de montrer la position cou
rante

1040 info$=CHR$( 47)

1050 LOCATE 1, 24

1060 PRINT info$;

1070 menux=5: menuy=24

1079 REM on definie le triangle

1080 SYMBOL 240, 0, 2, 6, 10, 18, 34, 66, 254

1090 SYMBOL 241, 255, 129, 129, 129, 129, 129,
129, 255

1099 REM exemples de couleurs de PEN 0 a
 7
```

```
1100 LOCATE menux,menuy

1110 PRINT CHR$(241);

1120 FOR colour=1 TO 7

1130 PEN colour

1140 PRINT CHR$(143);

1150 NEXT

1160 PEN 1

1169 REM affichage de symboles

1170 PRINT CHR$(47)CHR$(79)CHR$(232)CHR$
(240)CHR$(241);

1180 PRINT CHR$(23)CHR$(1);

1900 RETURN

1999 REM on positionne le curseur

2000 GOSUB 3000

2009 REM on repete jusqu'a la frappe de
'f'

2010 WHILE reponse$<>"f"

2019 REM effacement du curseur

2020 GOSUB 3000

2029 REM scrutation du clavier

2030 GOSUB 4000

2040 GOSUB 3000

2900 WEND

2910 RETURN

2999 REM routine affichage/effacement de
 lignes

3000 PLOT x,y,foregroundcolour

3010 IF linedraw=0 THEN RETURN
```

```
3020 DRAW startx, starty

3030 RETURN

3999 REM routine de lecture du clavier

4000 reponse$=LOWER$(INKEY$)

4010 IF reponse$="a" THEN y=y+2

4020 IF reponse$="z" THEN y=y-2

4030 IF reponse$="," THEN x=x-4

4040 IF reponse$="." THEN x=x+4

4048 REM la frappe de la barre d'espacem
ent positionne les points

4049 REM

4050 IF reponse$=" " AND y>31 THEN GOSUB
 5000: linedraw=foregroundcolour

4060 IF reponse$=" " AND y<32 THEN GOSUB
 6000

4070 IF reponse$="1" THEN IF linedraw=0
THEN linedraw=foregroundcolour ELSE line
draw=0

4900 RETURN

4999 REM

5000 PRINT CHR$(23)CHR$(0);

5010 GOSUB 3000

5020 PRINT CHR$(23)CHR$(1);

5030 startx=x: starty=y

5900 RETURN

5999 REM choix, rejet si non conforme

6000 IF x<128 OR x>543 OR y<16 THEN RETU
RN

6010 SOUND 7,400
```

```
6019 REM xcoord<384 montre qu'un changem
ent de couleur est necessaire

6020 IF x<384 THEN foregroundcolour=TEST
(x,y):GOSUB 7000:RETURN

6029 REM on verifie les coordonnees de x
 et on deduit le choix

6030 IF x<416 THEN info$=CHR$(47):GOSUB
7000:RETURN

6040 IF x<448 THEN info$=CHR$(79):GOSUB
7000:RETURN

6050 IF x<480 THEN info$=CHR$(232):GOSUB
 7000:RETURN

6060 IF x<512 THEN info$=CHR$(240):GOSUB
 7000:RETURN

6070 GOSUB 15000

6080 RETURN

7000 PEN foregroundcolour

7010 LOCATE 1,24

7020 PRINT info$

7030 RETURN

15000 REM a bientot

15010 RETURN
```

Ce programme permet de déplacer le curseur et d'effectuer des choix dans le menu, mais pour l'instant, seule l'option changement de couleur fonctionne.

Un choix de menu est indiqué si la coordonnée y du curseur est inférieure à 32 quand un point est fixé. Les lignes 4050 et 4060 pourraient être combinées en une seule instruction IF...THEN...ELSE. Elles sont présentées séparément pour plus de clarté. Le sous-programme de la ligne 6000 vérifie si la coordonnée x se trouve à l'intérieur de la zone de menu, puis il enregistre le choix effectué en fonction de cette coordonnée. Le sous-programme de la ligne 7000 affiche le

caractère INFO$ dans l'angle inférieur gauche de l'écran ; cette variable de chaîne est modifiée à chaque nouvelle sélection, pour la mise à jour.

L'adjonction de formes standard est relativement simple, mais il faut étudier avec attention la mise en œuvre de ces choix. Le tracé de cercles n'est pas très rapide, comme nous l'avons vu dans le Chapitre 3. Il est donc déconseillé de dessiner et d'effacer continuellement un cercle à l'aide d'une instruction EXOR, car le programme serait beaucoup trop lent. Une fois l'option sélectionnée, l'ordinateur opère de la façon suivante : le premier point fixé est considéré comme centre du cercle ; le point suivant est un point de la circonférence ; le rayon du cercle est donc égal à la distance séparant les deux points.

Le dessin d'un triangle ou d'un rectangle à l'aide d'une instruction EXOR est possible, car il ne concerne que peu de lignes, mais les deux options ont un fonctionnement légèrement différent. L'option rectangle nécessite la définition de deux points, alors que celle du triangle requiert la définition de trois points. Dans le cas d'un tracé de rectangle, sa position dépend de la diagonale tracée entre le premier point fixé et le sommet opposé.

Dans ces deux options, on utilise le déplacement du curseur et plusieurs lignes doivent être dessinées ou effacées simultanément. Ces options doivent donc être représentées par des sous-programmes séparés qui font appel au sous-programme de déplacement principal, mais pas au sous-programme de dessin et d'effacement de lignes utilisé dans le reste du programme.

```
4999 REM routine de trace cercle/rectang
le/triangle

5000 PRINT CHR$(23)CHR$(0);

5001 IF circle>0 THEN GOSUB 8000

5002 IF rectangle>0 THEN GOSUB 9000

5003 IF triangle>0 THEN GOSUB 10000

5010 GOSUB 3000

5020 PRINT CHR$(23)CHR$(1);

5030 startx=x: starty=y
```

```
5900 RETURN

5999 REM rejet du choix si non conforme

6000 IF x<128 OR x>543 OR y<16 THEN RETU
RN

6010 SOUND 7,400

6019 REM xcoord<384 montre qu'un changem
ent de couleur est demande

6020 IF x<384 THEN foregroundcolour=TEST
(x,y):GOSUB 7000:RETURN

6029 REM on verifie la coordonnee de x p
our deduire le choix

6030 IF x<416 THEN info$=CHR$(47):GOSUB
7000:RETURN

6040 IF x<448 THEN info$=CHR$(79):GOSUB
7000:circle=1:RETURN

6050 IF x<480 THEN info$=CHR$(232):GOSUB
 7000:rectangle=1:RETURN

6060 IF x<512 THEN info$=CHR$(240):GOSUB
 7000:triangle=1:RETURN

6070 GOSUB 15000

6080 RETURN

7000 PEN foregroundcolour

7010 LOCATE 1,24

7020 PRINT info$

7025 IF x<384 THEN RETURN

7029 REM nouveau choix

7030 circle=0

7040 triangle=0

7050 rectangle=0

7060 RETURN
```

```
7999 REM routine de trace du cercle

8000 IF circle=1 THEN circle=2: RETURN

8009 REM

8010 xd=ABS( x-startx): yd=ABS( y-starty)

8020 radius=SQR( xd*xd+yd*yd)

8030 MOVE startx, starty+radius

8040 FOR angle=0 TO 2*PI STEP PI/60

8050 DRAW startx+radius*SIN( angle), start
y+radius*COS( angle)

8060 NEXT

8070 DRAW startx, starty+radius

8080 PLOT startx, starty, 0

8090 circle=1

8100 RETURN

8999 REM routine de trace du rectangle

9000 IF rectangle=1 THEN rectangle=2: RET
URN

9010 MOVE startx, starty

9020 DRAWR x-startx, 0, foregroundcolour

9030 DRAWR 0, y-starty

9040 DRAWR startx-x, 0

9050 DRAWR 0, starty-y

9060 rectangle=1

9070 RETURN

9999 REM routine de trace du triangle

10000 IF triangle=1 THEN triangle=2: x1=x
: y1=y: RETURN
```

```
10010 IF triangle=2 THEN triangle=3: RETU
RN

10020 MOVE x, y

10030 DRAW startx, starty, foregroundcolou
r

10040 DRAW x1, y1

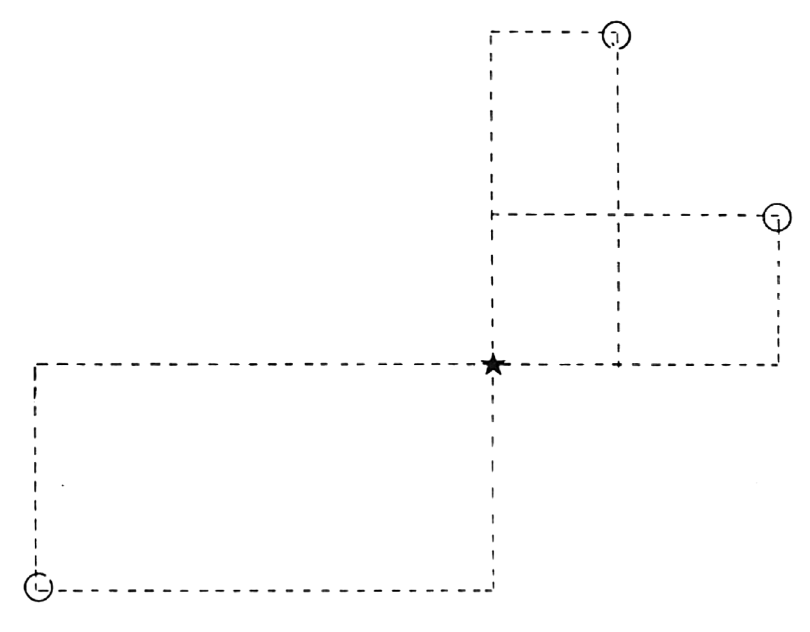
10050 DRAW x, y

10060 triangle=1

10070 RETURN

15000 REM a bientot

15010 RETURN
```



* – point fixé
○ – positions possibles pour le sommet opposé diagonalement
- - - – lignes non fixées

Figure 6.2 : On fixe la position d'un rectangle en déterminant seulement deux de ses sommets.

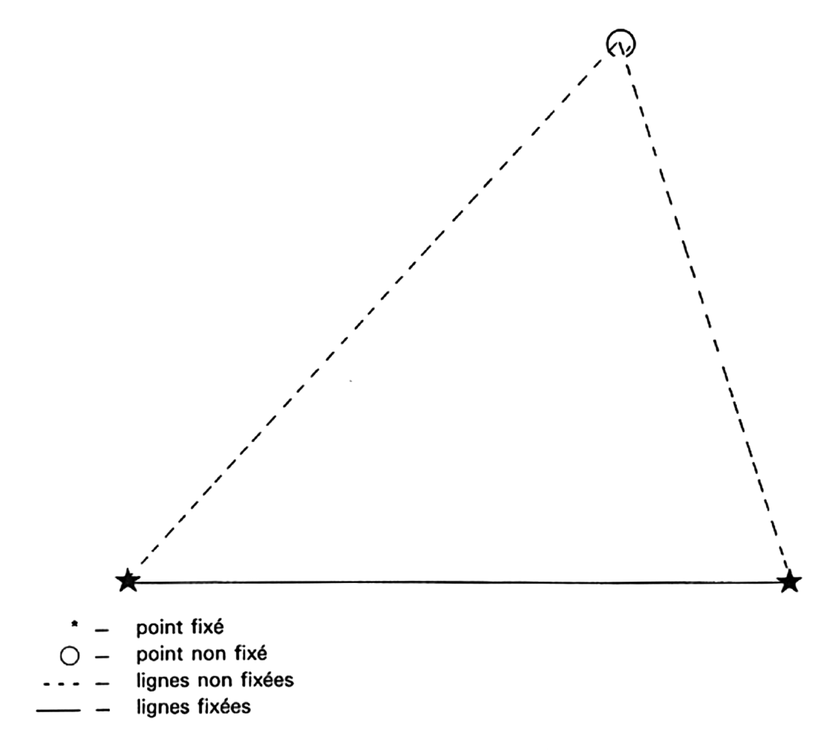


Figure 6.3 : Avant la définition du troisième sommet d'un triangle, deux de ses côtés ont une position indéterminée.

La sélection de l'option dessin de cercle, de rectangle ou de triangle est indiquée par le chiffre 1. Le sous-programme de la ligne 7000 est étendu pour que la sélection d'une option quelconque mette les indicateurs des autres options à 0 ; sinon, l'ordinateur pourrait essayer de dessiner en même temps un triangle, un rectangle et un cercle. La ligne 7025 permet de ne pas réinitialiser les indicateurs si le choix du menu ne concerne que la modification de la couleur.

Étant donné que les trois options de dessin de figures ne seront mises en œuvre qu'une fois fixé le nombre correct de points, la vérification de ces options est placée à l'intérieur du sous-programme 5000 qui fixe les points. Si l'indicateur d'une option particulière est supérieur à 0, le sous-programme lui correspondant est appelé lignes 5001 à 5003.

Les sous-programmes de dessin des lignes 8000, 9000 et 10000 ont une chose en commun : si le nombre de points fixé depuis la sélection de l'option n'est pas assez grand, la valeur de l'indicateur est

augmentée de 1 pour indiquer un autre point fixé, puis le sous-programme se termine, lignes 8000, 9000, 10000 et 10010. En fait, l'indicateur est également utilisé comme compteur pour montrer combien de points ont déjà été fixés. Il faut fixer deux points pour les options cercle et rectangle et trois points pour l'option triangle. Après cela, le sous-programme dessine la figure et donne à l'indicateur la valeur 1. Cela signifie par exemple, si l'option dessin de cercle a été sélectionnée, que le cercle continuera à être dessiné jusqu'à ce qu'une autre option soit choisie.

La structure du programme permet d'ajouter au menu d'autres options de dessin de figures, pour dessiner par exemple des ellipses ou des losanges.

EXERCICES

1. Étendre la gamme des couleurs affichées par le menu à dix couleurs.
2. Introduire dans le menu une commande d'effacement d'écran. Celle-ci attribue à l'ensemble de la zone graphique la couleur du premier plan.
3. Ajouter une nouvelle option de dessin de figures permettant de dessiner des arcs de cercle. Il faut définir trois points, le centre du cercle auquel appartient cet arc ainsi que le début et la fin de l'arc.

COLORATION DES FIGURES

Nous arrivons maintenant au sous-programme de coloration des figures. L'Amstrad possède une commande qui permet de remplir des zones de dessin avec une couleur déterminée. C'est la commande WINDOW, mais malheureusement, elle ne sert qu'avec les coordonnées texte. Par conséquent, il nous faut concevoir notre propre méthode de coloration. Pour être sûr de colorer l'ensemble d'une figure, il faut considérer chacun de ses points. Nous allons procéder par étapes en concevant un sous-programme qui colorera chaque point d'une ligne puis en l'étendant.

On prend un point arbitraire à l'intérieur de la figure. Il est possible de colorer tous les points qui se trouvent sur la même ligne en procédant en deux temps. Tout d'abord, le point placé à gauche de la

position en cours est examiné à l'aide de la commande TEST. Si celui-ci est affiché dans la couleur du fond, il est tracé dans la couleur du dessin. Ce point devient alors le nouveau "point de départ" et le point placé à sa gauche est examiné à son tour. Le processus est répété jusqu'à ce qu'un point d'une couleur différente de la couleur du fond soit rencontré ; ce point (voir la Figure 6.4) correspond à un point du contour de la figure.

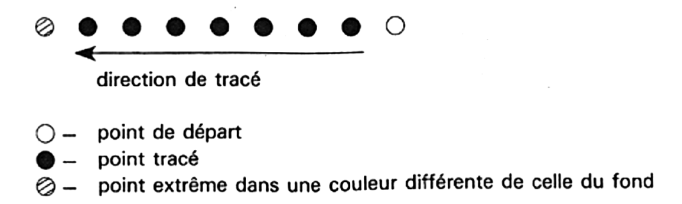


Figure 6.4

Effectivement, une ligne a été tracée à partir du point initial vers la gauche et jusqu'au contour de la figure. Les autres points de la ligne peuvent être colorés en reprenant le processus au point de départ, mais en examinant cette fois les points placés à droite. Le sous-programme suivant utilise cette procédure :

```
1 REM ce programme peut etre incorporer
au

2 REM programme principal et fonctiooner
 comme

3 REM un sous-programme

10 MODE 0

19 REM dessinez votre figure ici - un tr
iangle

20 MOVE 200,100

30 DRAW 450,350

40 DRAW 340,400

50 DRAW 200,100
```

```
54 REM on place 10 points au hasard

55 FOR count=1 TO 10

60 rand=INT(RND(1)*230):xhere=210+rand:y
here=110+rand

70 foregroundcolour=1:yfill=yhere

79 REM les points a gauche

80 xinc=-4:xfill=xhere

90 GOSUB 18000

98 REM maintenant ceux de droite

100 xinc=4:xfill=xhere-4

110 GOSUB 18000

120 NEXT

130 END

18000 t=0:WHILE t=0

18010 xfill=xfill+xinc

18020 t=TEST(xfill,yfill)

18028 REM si t=0 le point est dans la co
uleur du fond

18029 REM et doit etre affiche

18030 IF t=0 THEN PLOT xfill,yfill,foreg
roundcolour

18040 WEND

18050 RETURN
```

Le sous-programme de la ligne 18000 est appelé deux fois avec des paramètres différents pour les coordonnées x : initialement, l'incrément est −4 (examen des points placés à gauche) puis +4 (examen des points placés à droite). On remarque que cet incrément varie en fonction du mode ; la résolution en mode 1 est supérieure et un incrément de + ou −2 est nécessaire.

Que se passe-t-il si le point se trouve en dehors de la figure ? Malheureusement, la commande TEST n'est pas très utile dans ce cas et une des autres possibilités de l'Amstrad complique les choses ; si le point initial ne se trouve pas à l'intérieur d'une figure fermée, l'Amstrad continue à examiner des points placés à gauche, même s'ils sont en dehors de l'écran. La commande TEST enregistre un point placé en dehors de l'écran comme étant affiché dans la couleur du fond et l'ordinateur considère l'examen des points comme n'étant pas terminé ; il continue à prendre 4 comme incrément de déplacement sur la ligne, quelle que soit la valeur de x.

Il existe plusieurs façons de remédier à ce problème. On peut inclure un test de la valeur de xcoul :

```
59 REM point place aleatoirement mais en
 dehors du triangle

18000 t=0: WHILE t=0 AND xfill>0 AND xfil
l<639
```

Évidemment, cela ralentit le programme, car la coordonnée x de chaque point doit être testée. Il existe une autre possibilité plus rapide, qui consiste à dessiner une figure près du bord de l'écran ; l'ordinateur arrête de tracer des points lorsqu'il atteint ce bord, puisque le point suivant apparaît dans une couleur différente de celle du fond.

Le programme permettant de colorer des lignes uniques est complet, nous allons l'améliorer pour qu'il puisse colorer une figure. Il existe une méthode qui examine les points placés immédiatement au-dessus et au-dessous de la ligne qui vient d'être colorée. Si un point est affiché dans la couleur du fond, ses coordonnées peuvent être stockées dans un tableau pour qu'il serve de point de départ d'une ligne dessinée ultérieurement (voir la Figure 6.5).

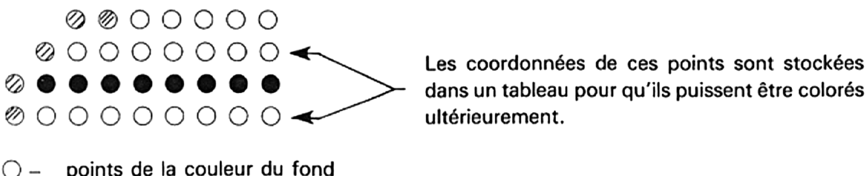


Figure 6.5

On a l'impression qu'il suffit de tester les points jusqu'à en trouver un de la couleur du dessin, mais certaines formes nécessitent le test de chaque point comme dans la Figure 6.6. Seules certaines parties verticales de la figure seront colorées, sauf si chaque point situé au-dessus de la ligne a été vérifié. Ces vérifications sont assez longues et il faut faire un choix entre l'efficacité et la perfection : est-il préférable d'utiliser un programme de coloration rapide mais imparfait, ou un programme plus lent qui colore tous les points de la forme quelle qu'elle soit ?

Un programme imparfait qui colore cependant avec succès de nombreuses formes nécessite seulement la vérification de quatre points.

```
10 MODE 0

14 REM on dimensionne un tableau pour le
s coordonnees des points non colores

15 DIM x(300), y(300)

19 REM on dessine la figure

20 startx=150: starty=150

30 MOVE startx, starty

40 DRAWR 100,0

50 DRAWR 0,100

60 DRAWR -100,0

70 DRAWR 0,-100

78 REM x et y sont les coordonnees d'un
point

80 x=200: y=200

90 foregroundcolour=1

100 GOSUB 15000

110 PRINT CHR$(23)CHR$(0);

999 END

15000 begin=2: finish=1
```

```
15010 xfill=x: yfill=y

15020 x(2)=x: y(2)=y

15030 PRINT CHR$(23)CHR$(0);

15040 PLOT x,y,0

15050 GOSUB 16000

15060 GOSUB 17000

15070 PRINT CHR$(23)CHR$(1);

15080 IF circle>0 OR rectangle>0 OR tria
ngle>0 THEN PLOT x,y,foregroundcolour

15090 RETURN

15999 REM on verifie la couleur du point

16000 xhere=xfill: yhere=yfill

16010 IF TEST(xfill,yfill)<>0 THEN RETUR
N

16020 xinc=-4

16029 REM

16030 GOSUB 18000

16040 xfill=xhere-4: yfill=yhere

16050 xinc=4

16060 GOSUB 18000

16070 RETURN

16999 REM creation d'une liste pour y me
ttre les points

17000 WHILE begin<>(finish+1)MOD 300

17010 xfill=x(begin): yfill=y(begin)

17020 begin=(begin+1)MOD 300

17030 GOSUB 16000

17040 WEND
```

```
17050 RETURN

18000 t=0:WHILE t=0

18010 xfill=xfill+xinc

18020 t=TEST(xfill,yfill)

18030 IF t=0 THEN PLOT xfill,yfill,foreg
roundcolour

18040 WEND

18050 xfill=xfill-xinc:yfill=yfill-2

18060 IF TEST(xfill,yfill)=0 THEN GOSUB
19000

18070 yfill=yfill+4

18080 IF TEST(xfill,yfill)=0 THEN GOSUB
19000

18090 RETURN

18999 REM sauvegarde des coordonnes de p
oints non colories afin de les colories
ulterieurement

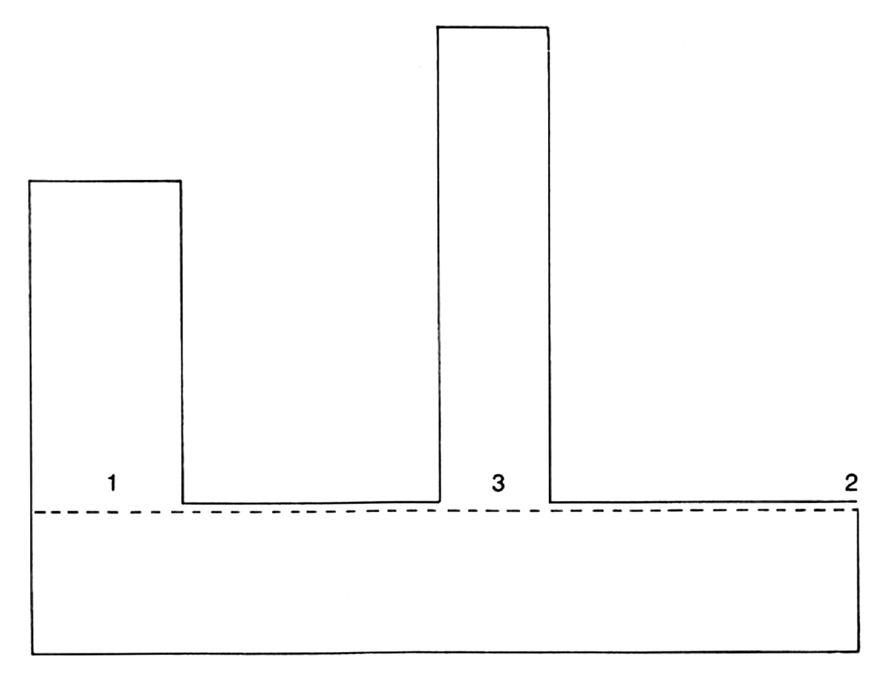
19000 finish=(finish+1)MOD 300

19010 x(finish)=xfill:y(finish)=yfill

19020 RETURN
```

On fait tourner le programme avec différentes formes. Il n'examine que les points placés diagonalement au-dessus et au-dessous de la ligne. Étant donné qu'un certain nombre de lignes seront colorées, les coordonnées de leurs extrémités sont sauvegardées dans les tableaux x() et y(). Deux pointeurs, ''début'' et ''fin'', indiquent les positions dans le tableau ; ''début'' donne le nombre d'éléments du tableau contenant les coordonnées (x,y) du point suivant à examiner, et ''fin'' donne le nombre d'éléments du tableau ''libres'' dans lesquels les coordonnées des nouveaux points non colorés peuvent être stockées.

Par exemple, après le dessin de la première ligne, il est vraisemblable que les quatre points placés au-dessus et au-dessous de cette



— — — Points les plus récents.

1 — Points placés au-dessus de la ligne qui seront colorés.

2 — Points placés au-dessus de la ligne qui sont déjà colorés ; la vérification n'ira pas plus loin.

3 — Points non colorés et qui le resteront, sauf si chaque point placé au-dessus de la ligne est testé.

Figure 6.6

ligne apparaîtront dans la couleur du fond ; leurs coordonnées seront donc stockées dans les tableaux x() et y(). Le sous-programme de la ligne 17000 examine successivement chaque point dont les coordonnées se trouvent dans le tableau et celui de la ligne 16000 établit les coordonnées des points placés de part et d'autre. Les points sont colorés dans le sous-programme de la ligne 18000 jusqu'à ce qu'un point d'une couleur différente de celle du fond soit rencontré. Ensuite, les lignes 18050 et 18080 vérifient la couleur des points placés diagonalement au-dessus et au-dessous du dernier point coloré et le sous-programme de la ligne 19000 les stocke s'ils ont pour couleur la couleur du fond.

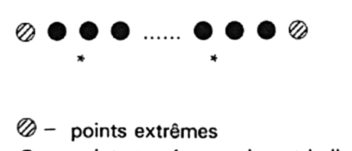


Figure 6.7

Les lignes 17000 et 19000 contiennent une instruction MOD, car les tableaux sont traités de façon circulaire. Évidemment, il est impossible de connaître le nombre de coordonnées que l'ordinateur doit stocker, mais le nombre 300 semble raisonnable. Une figure importante et compliquée peut cependant en nécessiter plus ; mais nous pouvons éviter d'établir un tableau plus grand en réutilisant les coordonnées précédentes (cela n'entraîne pas de perte de données car les éléments du tableau inférieur ont déjà été examinés auparavant).

Le sous-programme colore 90 % d'un cercle, mais il est inefficace quand l'arc contient des petites lignes horizontales. Celles-ci sont examinées et reconnues comme des points d'une couleur différente de celle du fond : le programme abandonne le dessin, car il "croit" être arrivé aux limites de la figure. Nous pouvons remédier à ce problème sans entraîner une trop grande perte de vitesse en examinant deux points supplémentaires.

```
15 REM trace d' un rectangle
20 startx=150: starty=150
30 MOVE startx, starty
40 DRAWR 100, 0
50 DRAWR 0, 100
55 DRAWR -10, 0
56 DRAWR 0, 50
57 DRAWR -60, 0
58 DRAWR 0, -50
60 DRAWR -30, 0
```

```
70 DRAWR 0,-100

16000 xhere=xfill: yhere=yfill

16010 IF TEST(xfill,yfill)<>0 THEN RETUR
N

16020 xinc=-4

16030 GOSUB 18000

16032 leftx=xfill

16040 xfill=yhere-4: yfill=yhere

16050 xinc=4

16060 GOSUB 18000

16062 rightx=xfill

16065 xfill=(leftx+rightx)/2: yfill=yhere
-2

16066 IF TEST(xfill,yfill)=0 THEN GOSUB
19000

16067 yfill=yhere+2

16068 IF TEST(xfill,yfill)=0 THEN GOSUB
19000

16070 RETURN
```

Ce programme colore des cercles avec succès, mais rencontre des difficultés avec certaines formes. La position du point initial détermine la taille de la zone colorée. Si un problème se pose, le reste de la zone peut être coloré à partir d'un nouveau point de départ. Par conséquent, le sous-programme semble offrir un bon compromis entre la vitesse et la perfection. Il peut être incorporé au programme principal de la manière suivante :

```
15 DIM x(300),y(300)

5031 IF fill=1 AND circle=0 AND rectangl
e=0 AND triangle=0 THEN GOSUB 15000

6070 IF fill=1 THEN fill=0:fill$=CHR$(24
1) ELSE fill=1:fill$=CHR$(233)
```

```
6080 LOCATE 17,24

6090 PRINT fill$;

6100 RETURN

8091 IF fill=1 THEN x=startx: y=starty: GO
SUB 15000

9061 IF fill=1 THEN x=(x+startx)/2: y=(y+
starty)/2: GOSUB 15000

10061 IF fill=1 THEN x=(x+startx+x1)/3: y
=(y+starty+y1)/3: GOSUB 15000
```

La coloration est mise en œuvre à partir du point fixé après le choix de l'option. Elle peut opérer en même temps que les options tracé de cercle, de rectangle ou de triangle ; dans ce cas, la figure est dessinée et automatiquement colorée par l'ordinateur à partir d'un point choisi lignes 8091, 9061 et 10061. L'autre possibilité consiste à utiliser cette option pour colorer une autre forme ; ceci est réalisé ligne 5031. On remarque qu'il faut sortir de l'option de coloration pour pouvoir dessiner des lignes, sinon le programme interprète le choix d'un point comme étant la position de départ de la coloration et essaie de colorer la totalité de l'écran.

EXERCICES

1. Étant donné que les points sont tracés individuellement dans le sous-programme de coloration, il est facile d'employer des combinaisons de couleurs pour remplir une figure (ligne 18030). Ajouter une option dans le menu qui permette de colorer en deux couleurs sélectionnées.
2. Le programme comporte un défaut ; la coloration s'arrête dès qu'un point d'une couleur différente de celle du fond est rencontré. Cela rend impossible la coloration d'une forme qui a été tracée sur une zone déjà colorée (par exemple, une porte bleue ne peut pas être dessinée sur un mur rouge). Modifier le programme pour que cette option colore n'importe quelle figure indépendamment des autres couleurs présentes.
3. Améliorer l'algorithme de coloration pour qu'il soit performant quelle que soit la forme de la figure.

SAUVEGARDE DE NOTRE RÉALISATION

Dans le Chapitre 4, nous avons stocké les coordonnées de points

et de lignes dans plusieurs tableaux qui ont été sauvegardés dans un fichier. Une démarche similaire peut être appliquée, mais le programme doit subir quelques modifications. Il nous faut savoir si les points étaient utilisés dans les options de tracé de cercles ou de rectangles et si les figures sont colorées ou non.

Une autre possibilité consiste à sauvegarder une copie de l'écran qui peut être chargée à nouveau pour visualisation ou modification à l'aide des options du menu habituelles. Cette méthode présente l'avantage de pouvoir être utilisée sans qu'on ait à modifier le programme.

La sauvegarde de l'écran est un exemple de la facilité avec laquelle l'Amstrad conserve une copie d'une partie de la mémoire graphique. Comme nous l'avons vu précédemment, l'écran est une représentation d'une partie de la mémoire RAM. En sauvegardant la zone mémoire appropriée, nous stockons effectivement une copie de l'écran sur bande.

L'Amstrad a besoin de certaines informations pour cette sauvegarde : les points de départ de la zone à stocker et sa taille, en octets. Ces informations sont également sauvegardées pour un chargement ultérieur.

Les sous-programmes de chargement et de sauvegarde sont facilement incorporés au programme et sont appelés respectivement à partir du clavier par la frappe des touches "i" (input : entrée) et "o" (output : sortie) :

```
4080 IF reponse$="i" THEN GOSUB 11000
4090 IF reponse$="o" THEN GOSUB 12000
11000 WINDOW 1,20,24,25: PEN 1
11010 PRINT"pour charger une figure"
11020 INPUT"nom de la figure";picture$
11030 LOAD picture$
11040 CLS
11050 WINDOW 1,20,1,25
11060 GOSUB 1000
11070 RETURN
```

```
12000 WINDOW 1,20,24,25: PEN 1

12010 PRINT"pour sauvegarder une figure"

12020 INPUT"tapez le nom de la figure";p
icture$

12030 SAVE picture$,B,&C000,&3FCF

12040 CLS

12050 WINDOW 1,20,1,25

12060 GOSUB 1000

12070 RETURN
```

Le sous-programme de la ligne 12000 exécute la sauvegarde d'un fichier écran. Il est impératif que des messages ne soient pas superposés à l'image pendant la sauvegarde de l'écran ; une fenêtre est donc établie en bas de l'écran pour supprimer temporairement le menu. La ligne 12030 stocke l'image : B signale un fichier binaire (format requis pour la sauvegarde d'une zone mémoire), &C000 est l'adresse hexadécimale du début de la mémoire écran qui est constituée de &3FCF octets. Après le stockage du fichier, le menu réapparaît et le déroulement du programme continue.

Le sous-programme de la ligne 11000 charge une figure. Il est semblable au programme de sauvegarde, bien que la commande LOAD de la ligne 11030 ait un format beaucoup plus simple. Le chargement d'une figure est très curieux : l'image n'est pas élaborée à partir du haut vers le bas comme on pourrait le croire, mais comme une série de bandes indépendantes. Cela reflète la complexité de l'organisation de l'écran dans laquelle des adresses mémoire qui se suivent numériquement correspondent souvent à des zones discontinues.

EXERCICES

1. Ajouter une option "zoom" au menu pour que l'image puisse être dessinée à une échelle différente.
2. Ajouter un sous-programme qui permette d'entrer du texte à partir du clavier et de le positionner à un endroit quelconque de l'écran.
3. Modifier le programme de telle sorte que toutes les couleurs du mode 0 soient disponibles.

7

TRANSFORMATIONS

Dans les chapitres précédents, nous avons appris à déplacer les points et à les visualiser lorsqu'ils se trouvent à la position désirée sur l'écran. Cependant, nous pouvons avoir besoin de déplacer une figure complète et non un point unique, et ce mouvement concerne souvent plus d'un pixel. Dans de nombreuses situations, il est utile de pouvoir exécuter une série de transformations sur une figure.

Nous avons déjà étudié la plus simple des transformations, la *translation*, qui est le déplacement d'un point ou d'un groupe de points dans une direction donnée. Tous les contrôles à partir du clavier servant à déplacer un point ou un groupe de points vers le haut, vers le bas, vers la gauche ou vers la droite provoquent une translation d'un pixel dans la direction choisie. Il est facile de modifier le programme précédent pour que des figures entières puissent être transformées.

```
1 REM a partir du programme de dessin du
 CH 4

2 REM on ajoute ou on modifie les lignes
 suivantes

65 figure=0: moveflag=0: GOSUB 9000

2010 IF figure=0 THEN GOSUB 1000

2065 REM on affiche la figure en tapant
la barre d'espacement

2070 IF reponse$=" " THEN IF figure=0 TH
EN GOSUB 3000: linedraw=foregrouncolour E
LSE figure=0

2080 IF figure=1 THEN GOSUB 10000 ELSE G
OSUB 1000

2083 REM la figure est dessinee quand 'f
' est tapee

2085 IF reponse$="f" THEN GOSUB 1000: IF
moveflag=0 THEN GOSUB 11000: moveflag=1

2086 IF reponse$="f" THEN x=x(nooflines)
: oldx=x: y=y(nooflines): oldy=y: figure=1
```

```
8999 REM lecture des donnees

9000 READ nooflines

9010 FOR count1=1 TO nooflines

9020 READ x(count1),y(count1),l(count1)

9030 NEXT

9040 RETURN

9050 DATA 7,300,100,0,400,100,1,490,190,
1,400,280,1,300,280,1,210,190,1,300,100,
1

10000 IF oldx=x AND oldy=y THEN RETURN

10009 REM on efface l'ancienne figure

10010 GOSUB 11000

10020 changex=x-oldx:changey=y-oldy

10030 oldx=x:oldy=y

10039 REM mise a jour des coordonnees

10040 FOR loop=1 TO nooflines

10050 IF changex=0 THEN y(loop)=y(loop)+
changey ELSE x(loop)=x(loop)+changex

10060 NEXT

10069 REM on affiche la figure a la nouv
elle position

10070 GOSUB 11000

10080 RETURN

11000 FOR loop=2 TO nooflines

11010 IF l(loop)>0 THEN MOVE x(loop),y(l
oop):DRAW x(loop-1),y(loop-1),l(loop)

11020 NEXT

11030 RETURN
```

La translation est la modification la plus simple du programme, mais elle est non avenue dans la mesure où l'approche employée n'est pas applicable à toutes les autres transformations comme par exemple la rotation. Une méthode plus générale utilise les *matrices*.

TRANSFORMATION DE MATRICES

Il est possible d'agrandir, de réfléchir une figure ou de lui faire subir une rotation en multipliant les coordonnées de tous ses points par les éléments d'une matrice appropriée. Nous allons étudier rapidement la multiplication de matrices, bien que la compréhension de ce processus dépasse largement le cadre de cet ouvrage.

On peut multiplier deux matrices en multipliant les éléments de chaque rangée de la première matrice par ceux de chaque colonne de la seconde matrice (voir la Figure 7.1). Dans cet exemple, on obtient la matrice finale en ajoutant les résultats de la multiplication de chaque élément de la rangée de la première matrice par les éléments des colonnes de la seconde matrice. La première matrice peut contenir plusieurs rangées et plus de deux colonnes, mais cet exemple concerne la transformation d'un point unique avec deux coordonnées.

$$(3\ 5) \times \begin{pmatrix} 2 & 4 \\ 1 & 2 \end{pmatrix} = (3*2 + 5*1 \quad 3*4 + 5*2)$$

$$= (11\ 22)$$

Figure 7.1

Un certain nombre de matrices différentes peuvent être utilisées pour générer une rotation ou pour agrandir une figure. Étant donné que le résultat de la multiplication de matrices se présente sous la même forme que la matrice à deux éléments initiale, on peut continuer à transformer la figure obtenue en la multipliant par une autre matrice de transformation, comme dans la Figure 7.2. En fait, le processus peut être simplifié ; on obtient le même résultat en commençant par multiplier entre elles les deux matrices de transformation puis en utilisant la matrice obtenue pour exécuter la transformation du point initial comme dans la Figure 7.3.

$$(11\ 22)\begin{pmatrix}1 & 2\\ 3 & 2\end{pmatrix} = (11*1 + 22*3 \quad 11*2 + 22*2)$$

$$= (77\ 66)$$

Figure 7.2

$$\begin{pmatrix}2 & 4\\ 1 & 2\end{pmatrix}\begin{pmatrix}1 & 2\\ 3 & 2\end{pmatrix} = \begin{pmatrix}2*1 + 4*3 & 2*2 + 4*2\\ 1*1 + 2*3 & 1*2 + 2*2\end{pmatrix}$$

$$= \begin{pmatrix}14 & 12\\ 7 & 6\end{pmatrix}$$

$$(3\ 5)\begin{pmatrix}14 & 10\\ 7 & 2\end{pmatrix} = (3*14 + 5*7 \quad 3*12 + 5*6)$$

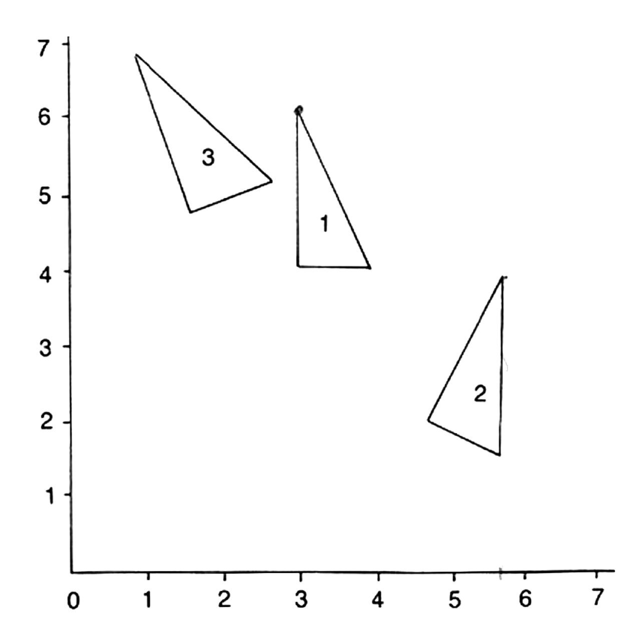
$$= (77\ 66)$$

Figure 7.3

Le principal inconvénient de l'utilisation de matrices de transformation 2*2 est qu'il est impossible de représenter la translation sous cette forme. Cela nous empêche d'adopter une méthode générale. La translation reste une transformation à part, alors que nous pourrions réduire à une seule matrice la série de matrices nécessaires à l'agrandissement, à la rotation et à la symétrie d'une figure. Il est possible d'étendre les matrices à 3*3, ce qui permet l'incorporation de la translation à l'aide d'une matrice appropriée. Étant donné que nous avons déjà étudié la translation, nous allons continuer à utiliser les matrices 2*2 et à traiter la translation comme une transformation particulière qui n'est pas soumise aux mêmes manipulations matricielles que les autres.

ROTATION

Deux matrices peuvent être utilisées pour faire exécuter une rotation à une figure dans le sens des aiguilles d'une montre et dans le sens inverse (voir la Figure 7.4). On voit, en comparant les résultats de la multiplication, que la seule différence réside dans les signes des éléments de la matrice. Si on définit la rotation dans le sens des aiguilles d'une montre en attribuant la valeur –1 à la variable "rota" et la rotation en sens inverse en lui attribuant la valeur 1, on a une



Le triangle 1 subit une rotation de 30° dans le sens des aiguilles d'une montre pour donner le triangle 2.

$$\begin{pmatrix} 3 & 4 \\ 4 & 4 \\ 4 & 6 \end{pmatrix} \begin{pmatrix} \cos 30 & -\sin 30 \\ \sin 30 & \cos 30 \end{pmatrix} = \begin{matrix} 4.6 & 2.0 \\ 5.5 & 1.5 \\ 5.6 & 3.7 \end{matrix}$$

Le triangle 1 subit une rotation de 20° dans le sens contraire des aiguilles d'une montre pour donner le triangle 3.

$$\begin{pmatrix} 3 & 4 \\ 4 & 4 \\ 3 & 6 \end{pmatrix} \begin{pmatrix} \cos 20 & \sin 20 \\ -\sin 20 & \cos 20 \end{pmatrix} = \begin{matrix} 1.5 & 4.8 \\ 2.4 & 5.1 \\ 0.8 & 6.7 \end{matrix}$$

Forme générale d'une rotation de θ degrés dans le sens des aiguilles d'une montre.

$$(x \ y) \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} = (x\cos\theta + y\sin\theta \quad -x\sin\theta + y\cos\theta)$$

Forme générale d'une rotation de θ degrés dans le sens contraire des aiguilles d'une montre.

$$(x \ y) \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} = (x\cos\theta + y\sin\theta \quad -x\sin\theta + y\cos\theta)$$

Figure 7.4

équation unique qui génère les coordonnées des rotations dans un sens quelconque, comme le montre la Figure 7.5.

Il est possible d'étendre le programme précédent qui permettait la translation d'une figure pour qu'il génère également la rotation :

```
2083 REM dessin de la figure quand 'f' e
st tapee

2084 REM rotation quand 't' est tapee

2085 IF reponse$="f" OR reponse$="t" THE
N GOSUB 1000: IF moveflag=0 THEN GOSUB 11
000: moveflag=1

2086 IF reponse$="f" OR reponse$="t" THE
N x=x(nooflines):oldx=x:y=y(nooflines):o
ldy=y: IF reponse$="f" THEN figure=1

2087 IF reponse$="t" THEN GOSUB 12000

12000 REM

12007 REM initialisation des cosinus et
des sinus

12008 REM rotation a 5 degres D'interval
le

12010 DEG

12020 transformx=COS(5)

12030 transformy1=COS(5)

12040 rotstop$=""

12049 REM la figure tourne j'usqu'a ce q
ue 't' soit tapee

12050 WHILE rotstop$<>"t"

12060 rotstop$=LOWER$(INKEY$)

12069 REM rotate=sens de rotation

12070 rotate=0

12079 REM 'a'=sens inverse 'c'=sens des
aiguilles d'une montre

12080 IF rotstop$="a" THEN rotate=-1
```

```
12090 IF rotstop$="c" THEN rotate=1

12100 IF rotate<>0 THEN GOSUB 11000: tran
sformy=SIN(5)*rotate: transformx1=-SIN(5)
*rotate: GOSUB 13000: GOSUB 11000

12110 WEND

12120 GOSUB 1000

12130 RETURN

13000 FOR loop=1 TO nooflines

13010 x1=x(loop)-centrex: y1=y(loop)-cent
rey

13020 x(loop)=transformx*x1+transformy*y
1+centrex

13030 y(loop)=transformx1*x1+transformy1
*y1+centrey

13040 NEXT

13050 RETURN
```

$$nouvx = vieux * \cos\theta + rota * vieuy * \sin\theta$$
$$nouvx = -rota * vieux * \sin\theta + vieuy * \sin\theta$$

Figure 7.5

On peut modifier le centre de rotation de la manière suivante :

```
12000 GOSUB 14000

14000 centre$=""

14010 GOSUB 1000

14019 REM on attend la frappe de 'f'

14020 WHILE centre$<>"f"

14030 GOSUB 1000
```

```
14040 centre$=LOWER$(INKEY$)

14049 REM deplacement du centre haut/bas
/gauche/droite

14050 IF centre$="a" THEN y=y+2

14060 IF centre$="z" THEN y=y-2

14070 IF centre$="," THEN x=x-4

14080 IF centre$="." THEN x=x+4

14090 GOSUB 1000

14100 WEND

14110 centrex=x: centrey=y: GOSUB 1000

14120 RETURN
```

AGRANDISSEMENT ET RÉDUCTION

Nous avons vu, dans un chapitre précédent, que l'agrandissement et la réduction d'une figure sont relativement faciles à réaliser. La méthode utilisant les matrices permet d'introduire certains agrandissements intéressants comprenant une variation du facteur d'échelle dans les directions x et y (voir la Figure 7.6). Nous allons ajouter cette possibilité à notre programme :

```
2077 REM dessin ou effacement figure

2080 IF figure=1 THEN GOSUB 10000 ELSE G
OSUB 1000

2082 REM modification de l'echelle quand
 's' est tapee

2083 REM dessin de la figure quand 'f' e
st tapee

2084 REM rotation quand 't' est tapee

2085 IF reponse$="f" OR reponse$="t" OR
reponse$="s" THEN GOSUB 1000: IF moveflag
=0 THEN GOSUB 11000: moveflag=1
```

```
2086 IF reponse$="f" OR reponse$="t" OR
reponse$="s" THEN x=x(nooflines):oldx=x:
y=y(nooflines):oldy=y:IF reponse$="f" TH
EN figure=1

2087 IF reponse$="t" THEN GOSUB 12000

2088 IF reponse$="s" THEN GOSUB 15000

15010 transformy=0

15020 transformx1=0

15030 scalestop$=""

15039 REM on modifie l'echelle jusqu'a l
a frappe de 's'

15040 WHILE scalestop$<>"s"

15050 scalestop$=LOWER$(INKEY$)

15060 transformx=0

15070 IF saclestop$="e" THEN transformx=
1.1:transformy1=1.1

15080 IF saclestop$="r" THEN transformx=
0.9:transformy1=0.9

15090 IF transformx<>0 THEN GOSUB 11000:
GOSUB 13000:GOSUB 11000

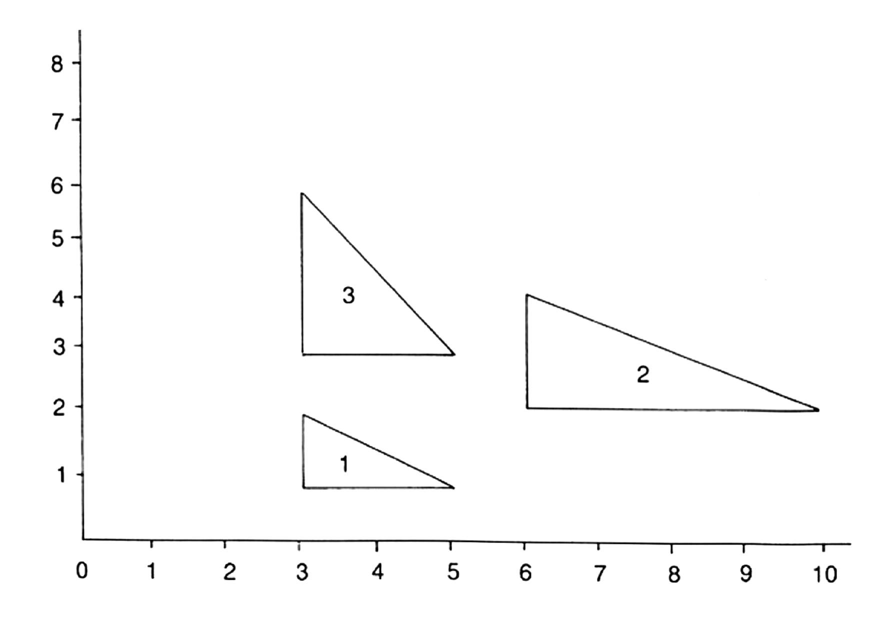
15100 WEND

15110 GOSUB 1000

15120 RETURN
```

Comme auparavant, il est préférable de pouvoir sélectionner le centre de l'agrandissement ou de la réduction :

```
15000 GOSUB 14000
```



L'agrandissement du triangle 1 est illustré par le triangle 2.

$$\begin{matrix}3&1\\5&1\end{matrix}\qquad\begin{pmatrix}2&0\\0&2\end{pmatrix}\qquad=\qquad\begin{matrix}6&2\\10&2\\6&4\end{matrix}$$

La modification de la diagonale conduit à un "étirement" parallèlement à un des axes.

$$\begin{pmatrix}3&1\\5&1\\3&2\end{pmatrix}\qquad\begin{pmatrix}1&0\\0&3\end{pmatrix}\qquad=\begin{pmatrix}3&3\\5&3\\3&6\end{pmatrix}$$

Forme générale d'un changement d'échelle ou d'un "étirement".

$$(x\ y)\ \begin{pmatrix}n1&0\\0&n2\end{pmatrix}=\quad(n1\times x\quad n2\times y)$$

Figure 7.6

SYMÉTRIE

La symétrie par rapport à l'abscisse ou à l'ordonnée peut être exécutée à l'aide des matrices de la Figure 7.7. Comme pour la rotation, les matrices diffèrent par le signe de certains de leurs éléments. La

réflexion n'est visible que si les axes sont déplacés, sinon le résultat est tracé en dehors de l'écran :

```
2081 REM image symetrique quand 'm' est
tapee

2082 REM modification echelle quand 's'
est tapee

2083 REM trace de la figure quand'f' est
 tapee

2084 REM rotation quand 't' est tapee

2085 IF reponse$="f" OR reponse$="t" OR
reponse$="s" OR reponse$="m" THEN GOSUB
1000: IF moveflag=0 THEN GOSUB 11000: move
flag=1

2086 IF reponse$="f" OR reponse$="t" OR
reponse$="s" OR reponse$="m" THEN x=x(no
oflines): oldx=x: y=y(nooflines): oldy=y: IF
 reponse$="f" THEN figure=1

2087 IF reponse$="t" THEN GOSUB 12000

2088 IF reponse$="s" THEN GOSUB 15000

15996 REM l'axe de reflexion est vertica
l ou horizontal

15997 REM on indique sa position en depl
acant le point puis en tapant x ou y

16000 GOSUB 14000

16010 axis$=""

16020 WHILE axis$<>"x" AND axis$<>"y"

16030 axis$=LOWER$(INKEY$)

16040 IF axis$="x" THEN transformx=1: tra
nsformy1=-1

16050 IF axis$="y" THEN transformx=-1: tr
ansformy1=1

16060 WEND

16070 GOSUB 11000: GOSUB 13000: GOSUB 1100
0
```

```
16080 GOSUB 1000

16090 RETURN
```

$$(x\ y)\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix} = (x\ -y) \quad \text{(symétrie sur l'axe des x)}$$

$$(x\ y)\begin{matrix}-1 & 0\\ 0 & 1\end{matrix} = (-x\ y) \quad \text{(symétrie sur l'axe des y)}$$

Figure 7.7

La symétrie par rapport à une ligne quelconque est plus complexe et nécessite d'autres transformations : par exemple, la translation d'un des axes, x ou y :

```
2034 REM deplacement de la figure quand
'p' est tapee

2085 IF reponse$="f" OR reponse$="t" OR
reponse$="s" OR reponse$="m" OR reponse$
="p" THEN GOSUB 1000:IF moveflag=0 THEN
GOSUB 11000:moveflag=1

2086 IF reponse$="f" OR reponse$="t" OR
reponse$="s" OR reponse$="m" OR reponse$
="p" THEN x=x(nooflines):oldx=x:y=y(noof
lines):oldy=y:IF reponse$="f" THEN figur
e=1

2087 IF reponse$="t" THEN GOSUB 12000

2088 IF reponse$="s" THEN GOSUB 15000

2089 IF reponse$="m" THEN GOSUB 16000

2090 IF reponse$="p" THEN GOSUB 17000

2095 WEND

17000 GOSUB 14000

17010 axis$=""

17020 transformx=1

17030 transformy1=1
```

```
17040 WHILE axis$<>"x" AND axis$<>"y"

17050 axis$=LOWER$(INKEY$)

17060 IF axis$="x" THEN transformx1=0: tr
ansformy=1

17070 IF axis$="y" THEN transformx1=1: tr
ansformy=0

17080 WEND
```

EXERCICES

1. Les transformations du programme sont toutes exécutées sur une figure dont les coordonnées sont lues à partir d'instructions DATA. Ajouter des sous-programmes au programme de transformation pour que l'on puisse dessiner une figure puis la transformer.
2. Modifier le programme pour inclure une option permettant de visualiser les positions précédentes de la figure pendant sa transformation.
3. Les différentes matrices de transformation peuvent également être employées pour générer des modèles. Nous avons déjà évoqué cela dans le Chapitre 4, lorsque nous avons étudié la rotation répétitive et l'agrandissement d'une figure. Écrire un programme qui permette de spécifier une ou plusieurs transformations à effectuer sur une figure. On peut également choisir le nombre de répétitions de la transformation sur la nouvelle figure dessinée. Une fois la transformation définie, l'ordinateur exécute sa tâche de façon répétitive et affiche les figures obtenues. Chaque figure peut être dessinée dans une couleur différente.
4. Certains programmes d'aide à la conception graphique contiennent des formes standard. L'utilisateur peut ''prendre'' une forme dans le menu, la déplacer à l'aide du curseur puis la ''fixer''. Écrire un programme de ce type qui facilite le dessin d'une maison. Inclure comme formes standard plusieurs types de portes et de fenêtres.

LA BIBLIOTHÈQUE SYBEX

OUVRAGES GÉNÉRAUX

VOTRE PREMIER ORDINATEUR *par Rodnay Zaks,*
296 pages, Réf. 394

VOTRE ORDINATEUR ET VOUS *par Rodnay Zaks,*
296 pages, Réf. 271

DU COMPOSANT AU SYSTÈME : une introduction aux microprocesseurs *par Rodnay Zaks,*
636 pages, Réf. 340

TECHNIQUES D'INTERFACE aux microprocesseurs *par Austin Lesea et Rodnay Zaks,*
450 pages, Réf. 339

LEXIQUE INTERNATIONAL MICRO-ORDINATEURS, avec dictionnaire abrégé en 10 langues
192 pages, Réf. 234

GUIDE DES MICRO-ORDINATEURS A MOINS 3 000 F *par Joël Poncet,*
144 pages, Réf. 322

LEXIQUE MICRO-INFORMATIQUE *par Pierre Le Beux,*
140 pages, Réf. 369

LA SOLUTION RS-232 *par Joe Campbell,*
208 pages, Réf. 0052

MINITEL ET MICRO-ORDINATEUR *par Pierrick Bourgault,*
198 pages, Réf. 0119

BASIC

VOTRE PREMIER PROGRAMME BASIC *par Rodnay Zaks,*
208 pages, Réf. 263

INTRODUCTION AU BASIC *par Pierre Le Beux,*
336 pages, Réf. 0035

LE BASIC PAR LA PRATIQUE : 60 exercices *par Jean-Pierre Lamoitier,*
252 pages, Réf. 0095

LE BASIC POUR L'ENTREPRISE *par Xuan Tung Bui,*
204 pages, Réf. 253

PROGRAMMES EN BASIC, Mathématiques, Statistiques, Informatique *par Alan R. Miller,*
318 pages, Réf. 259

BASIC, PROGRAMMATION STRUCTURÉE *par Richard Mateosian,*
352 pages, Réf. 429

JEUX D'ORDINATEUR EN BASIC *par David H. Ahl,*
192 pages, Réf. 246

NOUVEAUX JEUX D'ORDINATEUR EN BASIC *par David H. Ahl,*
204 pages, Réf. 247

FICHIERS EN BASIC *par Alan Simpson,*
256 pages, Réf. 0102

PASCAL

INTRODUCTION AU PASCAL *par Pierre Le Beux,*
496 pages, Réf. 330

LE PASCAL PAR LA PRATIQUE *par Pierre Le Beux et Henri Tavernier,*
562 pages, Réf. 361

LE GUIDE DU PASCAL *par Jacques Tiberghien,*
504 pages, Réf. 423

PROGRAMMES EN PASCAL pour Scientifiques et Ingénieurs *par Alan R. Miller,*
392 pages, Réf. 240

AUTRES LANGAGES

INTRODUCTION A ADA *par Pierre Le Beux,*
366 pages, Réf. 360

MICRO-ORDINATEURS

ALICE

JEUX EN BASIC POUR ALICE *par Pierre Monsaut,*
96 pages, Réf. 320

ALICE et ALICE 90, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 376

ALICE, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 401

ALICE, GUIDE DE L'UTILISATEUR *par Norbert Rimoux,*
208 pages, Réf. 378

ALICE, PROGRAMMATION EN ASSEMBLEUR *par Georges Fagot-Barraly,*
192 pages, Réf. 420

AMSTRAD

AMSTRAD, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 0105

AMSTRAD, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0107

AMSTRAD, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0108

AMSTRAD, PROGRAMMATION EN ASSEMBLEUR *par Georges Fagot-Barraly,*
208 pages, Réf. 0136

AMSTRAD EXPLORÉ *par John Braga,*
192 pages, Réf. 0135

APPLE / MACINTOSH

PROGRAMMEZ EN BASIC SUR APPLE II, Tomes 1 et 2 *par Léopold Laurent,*
208 pages, Réf. 333 et 380

APPLE II 66 PROGRAMMES BASIC *par Stanley R. Trost*,
192 pages, Réf. 283

JEUX EN PASCAL SUR APPLE
par Douglas Hergert et Joseph T. Kalash,
372 pages, Réf. 241

GUIDE DU BASIC APPLE II *par Douglas Hergert*,
272 pages, Réf. 0006

APPLE II, PREMIERS PROGRAMMES *par Rodnay Zaks*,
248 pages, Réf. 373

MACINTOSH, GUIDE DE L'UTILISATEUR
par Joseph Caggiano,
208 pages, Réf. 396

APPLE IIC, GUIDE DE L'UTILISATEUR
par Thomas Blackadar,
160 pages, Réf. 0089

MULTIPLAN SUR MACINTOSH
par Goulven Habasque,
240 pages, Réf. 0099

ATARI

JEUX EN BASIC SUR ATARI *par Paul Bunn*,
96 pages, Réf. 282

ATARI, PREMIERS PROGRAMMES *par Rodnay Zaks*,
248 pages, Réf. 387

ATARI, GUIDE DE L'UTILISATEUR *par Thomas Blackadar*,
192 pages, Réf. 354

ATMOS

JEUX EN BASIC SUR ATMOS *par Pierre Monsaut*,
96 pages, Réf. 346

ATMOS, 56 PROGRAMMES *par Stanley R. Trost*,
180 pages, Réf. 372

COMMODORE 64

JEUX EN BASIC SUR COMMODORE 64
par Pierre Monsaut,
96 pages, Réf. 0017

COMMODORE 64, PREMIERS PROGRAMMES
par Rodnay Zaks,
248 pages, Réf. 342

GUIDE DU BASIC VIC 20, COMMODORE 64
par Douglas Hergert,
240 pages, Réf. 312

COMMODORE 64, GUIDE DE L'UTILISATEUR
par J. Kascmer,
144 pages, Réf. 314

COMMODORE 64, 66 PROGRAMMES
par Stanley R. Trost,
192 pages, Réf. 319

COMMODORE 64, GUIDE DU GRAPHISME
par Charles Platt,
372 pages, Réf. 0053

COMMODORE 64, JEUX D'ACTION *par Eric Ravis*,
96 pages, Réf. 403

COMMODORE 64, 1ers CONTACTS
par Marty Dejonghe et Caroline Earhart,
208 pages, Réf. 390

COMMODORE 64, BASIC APPROFONDI
par Gary Lippman,
216 pages, Réf. 0100

DRAGON

JEUX EN BASIC SUR DRAGON *par Pierre Monsaut*,
96 pages, Réf. 324

EXL 100

EXL 100, JEUX D'ACTION *par Pierre Monsaut*,
96 pages, Réf. 0126

GOUPIL

PROGRAMMEZ VOS JEUX SUR GOUPIL
par François Abella,
208 pages, Réf. 264

HECTOR

HECTOR JEUX D'ACTION *par Pierre Monsaut*,
96 pages, Réf. 388

IBM

IBM PC EXERCICES EN BASIC *par Jean-Pierre Lamoitier*,
256 pages, Réf. 338

IBM PC GUIDE DE L'UTILISATEUR
par Joan Lasselle et Carol Ramsey,
160 pages, Réf. 301

IBM PC 66 PROGRAMMES BASIC *par Stanley R. Trost*,
192 pages, Réf. 359

GRAPHIQUES SUR IBM PC *par Nelson Ford*,
320 pages, Réf. 357

GUIDE DU PC DOS *par Richard A. King*,
240 pages, Réf. 0013

LASER

LASER JEUX D'ACTION *par Pierre Monsaut*,
96 pages, Réf. 371

MO 5

MO 5 JEUX D'ACTION *par Pierre Monsaut*,
96 pages, Réf. 0067

MO 5, PREMIERS PROGRAMMES *par Rodnay Zaks*,
248 pages, Réf. 370

MO 5, 56 PROGRAMMES *par Stanley R. Trost*,
160 pages, Réf. 375

MO 5, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
192 pages, Réf. 384

MO 5, DYNAMIQUE CINÉMATIQUE, MÉTHODE POUR LA PROGRAMMATION DES JEUX *par Daniel Lebigre*,
272 pages, Réf. 0118

MSX

MSX, JEUX D'ACTION *par Pierre Monsaut*,
96 pages, Réf. 411

MSX, INITIATION AU BASIC *par Rodnay Zaks,*
248 pages, Réf. 410

MSX, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0109

MSX, GUIDE DU GRAPHISME *par Mike Shaw,*
192 pages, Réf. 0132

ORIC

JEUX EN BASIC SUR ORIC *par Peter Shaw,*
96 pages, Réf. 278

ORIC PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 344

SHARP

DÉCOUVREZ LE SHARP PC-1500 ET LE TRS-80 PC-2
par Michel Lhoir,
2 tomes, Réf. 261-262

SPECTRAVIDEO

SPECTRAVIDEO, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 377

SPECTRUM

PROGRAMMEZ EN BASIC SUR SPECTRUM
par S.M. Gee,
208 pages, Réf. 252

JEUX EN BASIC SUR SPECTRUM *par Peter Shaw,*
96 pages, Réf. 276

SPECTRUM, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 381

SPECTRUM JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 368

TI 99/4

PROGRAMMEZ VOS JEUX SUR TI 99/4
par François Abella,
160 pages, Réf. 303

TO 7

JEUX EN BASIC SUR TO 7 *par Pierre Monsaut,*
96 pages, Réf. 0026

TO 7, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 328

TO 7, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot-Barraly,
192 pages, Réf. 350

JEUX SUR TO 7 et MO 5 *par Georges Fagot-Barraly,*
168 pages, Réf. 0134

GESTION DE FICHIERS SUR TO 7 ET MO 5
par Jean-Pierre Lhoir,
136 pages, Réf. 0127

TO 7, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 374

TRS-80

PROGRAMMEZ EN BASIC SUR TRS-80
par Léopold Laurent,
2 tomes, Réf. 366-251

JEUX EN BASIC SUR TRS-80 MC-10 *par Pierre Monsaut,*
96 pages, Réf. 323

JEUX EN BASIC SUR TRS-80 *par Chris Palmer,*
96 pages, Réf. 302

JEUX EN BASIC SUR TRS-80 COULEUR
par Pierre Monsaut,
96 pages, Réf. 325

TRS-80 MODÈLE 100, GUIDE DE L'UTILISATEUR
par Orson Kellog,
112 pages, Réf. 300

TRS-80 COULEUR, PREMIERS PROGRAMMES
par Rodnay Zaks,
248 pages, Réf. 414

TRS-80 COULEUR, 56 PROGRAMMES
par Stanley R. Trost,
160 pages, Réf. 413

VIC 20

PROGRAMMEZ EN BASIC SUR VIC 20
par G. O. Hamann,
2 tomes, Réf. 329-337

JEUX EN BASIC SUR VIC 20 *par Alastair Gourlay,*
96 pages, Réf. 277

VIC 20, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 341

VIC 20 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 345

VG 5000

VG 5000, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 422

VG 5000, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0128

ZX 81

ZX 81 GUIDE DE L'UTILISATEUR *par Douglas Hergert,*
208 pages, Réf. 351

ZX 81 56 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 304

GUIDE DU BASIC ZX 81 *par Douglas Hergert,*
204 pages, Réf. 285

JEUX EN BASIC SUR ZX 81 *par Mark Charlton,*
96 pages, Réf. 275

ZX 81 PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 343

MICROPROCESSEURS

PROGRAMMATION DU Z80 *par Rodnay Zaks,*
618 pages, Réf. 358

APPLICATIONS DU Z80 *par James W. Coffron,*
304 pages, Réf. 274

PROGRAMMATION DU 6502 *par Rodnay Zaks,*
376 pages, Réf. 0031, 2ème édition

APPLICATIONS DU 6502 *par Rodnay Zaks,*
288 pages, Réf. 332

PROGRAMMATION DU 6800
par Daniel Jean David et Rodnay Zaks,
374 pages, Réf. 327

PROGRAMMATION DU 6809
par Rodnay Zaks et William Labiak,
392 pages, Réf. 0139

PROGRAMMATION DU 8086/8088
par James W. Coffron,
304 pages, Réf. 0016

MISE EN OEUVRE DU 68000 *par C. Vieillefond,*
352 pages, Réf. 0133

ASSEMBLEUR DU 8086/8088
par François Retoreau,
616 pages, Réf. 0093

SYSTÈMES D'EXPLOITATION

GUIDE DU CP/M AVEC MP/M *par Rodnay Zaks,*
354 pages, Réf. 336

CP/M APPROFONDI *par Alan R. Miller,*
380 pages, Réf. 334

INTRODUCTION AU p-SYSTEM UCSD
par Charles W. Grant et Jon Butah,
308 pages, Réf. 365

GUIDE DE MS-DOS *par Richard A. King,*
360 pages, Réf. 0117

APPLICATIONS ET LOGICIELS

INTRODUCTION AU TRAITEMENT DE TEXTE
par Hal Glatzer,
228 pages, Réf. 243

INTRODUCTION A WORDSTAR *par Arthur Naiman,*
200 pages, Réf. 0062

WORDSTAR APPLICATIONS *par Julie Anne Arca,*
320 pages, Réf. 0005

VISICALC APPLICATIONS *par Stanley R. Trost,*
304 pages, Réf. 258

VISICALC POUR L'ENTREPRISE *par Dominique Helle,*
304 pages, Réf. 309

INTRODUCTION A dBASE II *par Alan Simpson,*
280 pages, Réf. 0064

DE VISICALC A VISI ON *par Jacques Bourdeu,*
256 pages, Réf. 321

MULTIPLAN POUR L'ENTREPRISE
par D. Helle et G. Boussand,
304 pages, Réf. 0079

dBASE II APPLICATIONS *par Christophe Stehly,*
248 pages, Réf. 416

INTRODUCTION A LOTUS 1-2-3
par Chris Gilbert et Laurie Williams,
272 pages, Réf. 0106

LOGISTAT, ANALYSE STATISTIQUE DES DONNÉES
par Fredj Tekaia et Michele Bidel,
192 pages, Réf. 0132

La plupart de ces ouvrages existent en version anglaise.

POUR UN CATALOGUE COMPLET DE NOS PUBLICATIONS

FRANCE
6-8, Impasse du Curé
75881 PARIS CEDEX 18
Tél. : (1) 203.95.95
Télex : 211801

U.S.A.
2344 Sixth Street
Berkeley, CA 94710
Tel. : (415) 848.8233
Telex : 336311

ALLEMAGNE
Vogelsanger. WEG 111
4000 Düsseldorf 30
Postfach N° 30.09.61
Tel. : (0211) 626441
Telex : 08588163



Paris • Berkeley • Düsseldorf

Achevé d'imprimer le 4 septembre 1985 sur les presses de l'Imprimerie «La Source d'Or»
63200 Marsat - Dépôt légal : 3e trimestre 1985 - Imprimeur n° 1832

L'Amstrad possède des qualités graphiques étonnantes et offre de nombreuses instructions permettant de réaliser des applications très performantes. Cet ouvrage présente à l'aide d'exemples de programmes en BASIC toutes les techniques indispensables pour permettre au lecteur de réaliser de façon simple et rapide ses programmes graphiques. Les exemples étudiés permettront entre autres de mieux comprendre les principes de gestion des couleurs, d'utilisation des différents modes vidéo, du tracé de courbes et de la réalisation de graphiques animés. Les programmes fonctionnent sur les modèles CPC 464, CPC 664 et CPC 6128.




AMSTRAD
CPC
OCR
MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA
ACME
https://acpc.me/